



Universiteit Gent  
Faculteit Wetenschappen  
Vakgroep Toegepaste Wiskunde en  
Informatica

Mining and Modelling Interaction Networks for Systems  
Biology

---

Timur Fayruzov

Supervisors:  
Prof. Dr. Véronique Hoste  
Dr. Chris Cornelis  
Prof. Dr. Martine De Cock

Dissertation submitted to the Faculty of Sciences of Ghent University  
in fulfillment of the requirements for the degree of Doctor of Computer Science

October 2010



# Contents

<b>Acknowledgements</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
<b>2 Machine learning for text mining</b>	<b>6</b>
2.1 Introduction . . . . .	6
2.2 Natural language processing . . . . .	8
2.2.1 Lexical processing . . . . .	8
2.2.2 Syntactic processing . . . . .	10
2.3 Biological corpora . . . . .	12
2.4 Machine learning . . . . .	15
2.4.1 Learning problems . . . . .	15
2.4.2 Support vector machines . . . . .	16
2.4.3 Performance metrics . . . . .	22
2.5 Conclusions . . . . .	24
<b>3 Feature analysis for PPI extraction from text</b>	<b>26</b>
3.1 Introduction . . . . .	26
3.2 Problem statement . . . . .	28
3.3 SVM with structured kernels . . . . .	31
3.4 Related work . . . . .	37
3.5 Results . . . . .	40
3.5.1 Experimental setup . . . . .	40
3.5.2 Discussion . . . . .	40
3.6 Conclusion . . . . .	50
<b>4 Systems biology and computer science</b>	<b>51</b>
4.1 Introduction . . . . .	51

---

4.2	Modelling approaches . . . . .	52
4.2.1	$\pi$ -calculus . . . . .	53
4.2.2	Petri Nets . . . . .	55
4.2.3	Pathway Logic . . . . .	57
4.3	Boolean networks . . . . .	58
4.3.1	General definitions . . . . .	59
4.3.2	Random Boolean Networks . . . . .	60
4.3.3	Generalized boolean networks . . . . .	61
4.3.4	Threshold boolean networks . . . . .	61
4.4	Conclusion . . . . .	62
<b>5</b>	<b>Modelling time-dependent systems with ASP</b>	<b>64</b>
5.1	Introduction . . . . .	64
5.2	Answer set programming . . . . .	66
5.3	Time-dependent programs . . . . .	70
5.4	Markovian programs . . . . .	77
5.5	Discovery of steady states . . . . .	89
5.6	Related work . . . . .	92
5.7	Conclusions . . . . .	96
<b>6</b>	<b>Modelling biological regulatory networks with ASP</b>	<b>97</b>
6.1	Introduction . . . . .	97
6.2	Related work . . . . .	98
6.3	Boolean networks and ASP . . . . .	102
6.4	Building a network model in ASP . . . . .	104
6.4.1	Basic framework . . . . .	104
6.4.2	Extended framework . . . . .	116
6.5	Network modelling and analysis algorithms . . . . .	123
6.5.1	State Space Building approach . . . . .	124
6.5.2	Querying networks . . . . .	126
6.6	Experimental validation . . . . .	130
6.6.1	Budding Yeast network . . . . .	131
6.6.2	Fission Yeast network . . . . .	134
6.6.3	Mammalian cell cycle network . . . . .	138
6.7	Conclusions . . . . .	146
<b>7</b>	<b>Systems for PPI retrieval and extraction</b>	<b>150</b>
7.1	Introduction . . . . .	150
7.2	Online tools for biologists . . . . .	151

---

7.3	Motivation for the development of PRISE . . . . .	155
<b>8</b>	<b>PRISE</b>	<b>158</b>
8.1	Introduction . . . . .	158
8.2	RapidMiner . . . . .	161
8.3	Backend architecture . . . . .	161
8.3.1	PPI extraction task . . . . .	161
8.3.2	Domain models . . . . .	162
8.3.3	Application mode . . . . .	165
8.3.4	Evaluation mode . . . . .	167
8.3.5	PRISE backend implementation . . . . .	168
8.4	Frontend architecture . . . . .	169
8.4.1	Server-side architecture . . . . .	169
8.4.2	Client-side architecture . . . . .	171
<b>9</b>	<b>Summary</b>	<b>177</b>
	<b>Bibliography</b>	<b>183</b>



# Acknowledgements

Doing a PhD is hard work, and finishing this thesis would not have been possible without the help of many people I have met in these four years. My biggest acknowledgement is to Martine. Your trust and support provided me with the power to accomplish this work. Your guidance gave me many bright ideas, and you always knew the right words in the moments of my desperation. I would like to say a separate thank you for your patience with proofreading my English. Chris, thank you for your help with shaping my research goals and with writing of papers. Veronique, your help on language processing was invaluable and meetings with you and with the LT3 team were always a cheerful experience. I also would like to thank Els, although it is not in this thesis, the Web People Search project was an interesting experience.

Another portion of appreciation goes to people in the University of Washington, Tacoma, who made my stay there both productive and enjoyable. Ankur, thank you for the endless source of ideas that you generously share with everyone. Richard, Nick, Seth, Aparna, your help with PRISE implementation was invaluable. Rinkesh, thank you for your company during my stays: our coffee breaks were great.

A special thanks goes to my UGent office mates Gilles and Nele for never saying a word against being distracted from their work with endless questions about life in Belgium. Nele, thank you for being so devoted to teach me Dutch. Jeroen, your ASP expertise was an invaluable help that made this thesis much better.

I am grateful to my parents that were willing to meet me at any reachable place on earth. Although we did not meet too often, I felt your support everywhere.

The last, but the most important gratitude goes to my family – Tanya and Masha. You have learned fully what it means to be a family of a PhD student, without you I would not be able to come to this point. Your love and support is filling my life with joy every day.

# 1 Introduction

---

Almost twenty years ago the World Wide Web started a new era in information search by bringing new ideas on how information should be organized and accessed. Information has never been as widely available as in our days - using a search engine one can get thousands of documents on almost any topic with a few keystrokes and a mouse click. Moreover, in the last decade the concept of the social web has emerged, which has created a new trend of producing information rather than simply consuming it. However, problems of managing this information come along with the many obvious benefits of the Web. Getting a lot of search results is great, however users are left with the burden of processing this data to find the pieces of information they really need.

The field of biology is a striking example of this problem. Biology takes full advantage of the information dissemination technologies provided by the Web. MEDLINE [102], is the most comprehensive database of biomedical article citations, is a great example of the new information age in biology. Moreover, hundreds of biological journals provide open access to their articles, many research groups and biological laboratories maintain freely available databases of biological experiments such as BIND [11], UniProt [166] or DIP [145] and biologists make use of the latest Web developments such as ontologies by building knowledge bases such as the Gene Ontology [6]. On the other hand, the abundance of information makes it increasingly difficult to browse information on specific topics and stay informed on the latest developments.

The immense growth of biological data is due to the development of new high-throughput experimental techniques that capture data about thousands of genes and proteins at once. With the development of such techniques in the biological community came the understanding that it is not possible anymore



to study a particular gene or protein in isolation, and that new system level methods are needed to comprehend the available data and build realistic models of living organisms. These factors gave rise to the new discipline of systems biology that emerges from biology, statistics and computer science domains to create instruments to build comprehensive models of biological processes. Systems biology tries to achieve this goal by attempting to create *in silico* models that reflect the behaviour of real living organisms [141].

This thesis addresses the problem of *information overload* in the biological domain. We study two aspects of this problem: information extraction and modelling. These two aspects are interconnected, as the ultimate goal of collecting the information is to build models, and building models is only possible when enough information is available.

We address the first aspect by studying methods for protein-protein interaction (PPI) extraction from text of biological articles. In the last ten years many approaches to handle this problem were proposed, each of them taking into account specific textual features. Modern text mining approaches may employ thousands of features induced from the text, striving to find the combination that would be the most useful for PPI extraction. These features, although all coming from text, can be classified into several groups, such as lexical (explicitly present in text), shallow syntactic (induced with lightweight natural language processing) and deep syntactic (induced with full parsing techniques) ones. Many efforts have been made to build a good PPI extraction system, little attention has been devoted to the study of the usefulness of the particular categories of features for this task. We address this problem by studying the impact of different feature types on the accuracy of PPI extraction methods.

To address the second aspect, namely the biological network modelling, we propose a logic-based modelling framework that allows us to construct and analyze biological regulatory networks. In the field of systems biology many approaches have been proposed to build biological models ranging from Boolean networks to differential equations. Each modelling formalism has advantages and drawbacks; usually the ability to incorporate many fine details in a model comes at a cost of scalability and modelling complexity. We follow an incremental strategy and start with the most simple boolean network representation and extend it by adding capabilities to capture complex relationships in an intuitive way. Moreover, we provide a querying mechanism that enriches the biologist's modelling experience and allows him to study a system's behaviour by posing complex queries to that system.

This thesis contains three parts. In the first part, we study methods for protein-protein interaction extraction from texts. Chapter 2 is an introduction

to the techniques that are employed in the PPI extraction task: natural language processing (NLP) and machine learning. NLP is a necessary prerequisite for any information extraction task as it introduces structure into the otherwise unstructured (from the information extraction point of view) textual data. NLP includes lexical, morphological and syntactic processing steps that are detailed in Chapter 2. Another component of many PPI extraction systems is a machine learning algorithm that is used to actually find the interactions. In our research, we have used the Support Vector Machines (SVM) algorithm, which is explained in Chapter 2. The essential component of an SVM is a kernel function, that can be modified to the needs of any particular classification task. In Chapter 3 we use non-standard convolution kernels, thus we provide the necessary background on this kernel type in Chapter 2 as well.

In Chapter 3, we study the influence of different feature types on the performance of PPI extraction methods. Different types of features can be induced from text, starting with the most obvious ones such as words themselves, word sequences ( $n$ -grams), orthographic and punctuation features to more elaborate features that do not explicitly occur in the text such as part-of-speech tags and grammatical relations between words in a sentence. In the quest of finding PPIs, researchers generate thousands of features hoping that these will include the ones that are the most indicative for the extraction task. However, little attention has been devoted to studying the impact of different feature types on the performance of the PPI extraction task. We divide features into three groups: lexical, shallow syntactic and deep syntactic features. We devise four SVM classifiers with different kernels that use these groups of features and analyze how elimination of certain feature types affects the performance. This work stems from our research papers ([52, 50] and [51]).

The second part of the thesis deals with the modelling aspect of systems biology. In Chapter 4, we provide an overview of formalisms proposed to deal with the task of modelling biological processes, and discuss their potential. Moreover, in this chapter we provide the formal description of Boolean networks and threshold boolean networks that are essential for understanding Chapter 6.

In Chapter 5, we employ Answer Set Programming (ASP) to present a theoretical framework for modelling systems that evolve with time. To this end, we introduce a notion of time-dependent programs. The usual problem that arises when analyzing these systems is to find their steady states. Traditional ASP solving approaches are suboptimal for finding steady states in scenarios where the number of time steps is not known in advance, which is usually the case in the real world. Therefore, we propose a theoretical framework that can be used to model temporal systems, and more in particular first-order Markovian systems,

where the next state depends only on the previous state and does not depend on any future states of the system. We flesh out the theoretical underpinnings of this framework and provide an algorithm that finds steady states in cases where the number of steps is not known in advance.

In Chapter 6, we use the Markovian programs proposed in Chapter 5 to build a framework for biological regulatory network modelling. We start by simulating boolean network semantics in an ASP setting. We extend this semantics with additional constructs that cannot be directly modelled with Boolean networks. Moreover, we devise a framework-specific algorithm for state space building. Another advantage of our framework is a more explicit model representation compared with the Boolean networks. Additionally, we employ the techniques developed in Chapter 5 to develop a querying mechanism that finds steady states and allows the use of queries for arbitrary states/trajectories that can be reached in the network, thus allowing to perform a fine-grained network analysis.

Part of the work presented in the second part of this thesis has been published ([53], [56] and [55]).

In the third part of the thesis we describe the implementation of PRISE – a **PR**otein Interaction Search Engine. Chapter 7 provides an overview of information retrieval and extraction systems for biological data. This chapter motivates the need for an integrated framework for interaction extraction methods development. Chapter 8 introduces a framework based on the RapidMiner machine learning library. We present the PRISE system, that contains this framework in the backend. PRISE combines the developments proposed in first two parts of this thesis in a single application that extracts protein-protein interactions from texts and can potentially use this information as a basis for building regulatory networks. These networks can be analyzed with the techniques proposed in the second part of the thesis. Moreover, the libraries constituting PRISE are designed in a modular way and provide a convenient workbench for the researchers and practitioners of PPI extraction methods. A part of the work, presented in this chapter has been summarized in our paper [54].

Overall, this thesis contributes to two important aspects of biological information processing: information extraction and reasoning. We believe that information management in the biological domain is a challenging area for computer science applications, and it can benefit from new developments in information extraction and knowledge representation and reasoning, such as those described in this dissertation.

# 2 Machine learning for text mining

---

## 2.1 Introduction

Biology has experienced a drastic increase in the rate of information production. Synthesizing information from different data sources has the great potential to build new hypotheses and come up with discoveries. However, this idealistic picture is often not seen in reality. The downside of the rich information flow is that it is often very loosely structured; for example when the search process returns the set of papers on a topic, a researcher still needs to read these papers in order to be able to use this information. Therefore, it is increasingly difficult to keep track of the new developments in a certain subfield and to consult existing knowledge.

Computer science has a long-standing interest in text understanding, information extraction and reasoning methods (see e.g. [155, 26]). More specifically, data and text mining techniques are seen as a potential solution to the problem of efficient information management. By data mining we mean knowledge extraction from structured sources such as databases. By analyzing correlations between data sets that were possibly never studied together before, data mining can aid in formulating new hypotheses that can be further evaluated by researchers. However, biologists are often confronted with unstructured data, presented as natural language articles on the topic. To deal with this type of information, text mining is used.

Text mining, as described by Hearst [74], is the process of extracting knowledge from unstructured textual data. The process of text mining includes three steps: information retrieval (IR) to find relevant texts, information extraction (IE) to derive a structural representation of a retrieved text and data mining that

finds new knowledge by analyzing the outcome of the IE step. It is important to distinguish between IR and IE. IR algorithms (many of which can be found in e.g. [98] or [31] and other proceedings of the SIGIR conference series) are typically designed to work on a scale of thousands or even millions of documents in order to find the ones that are most likely to answer an (often superficially defined) user query. IR provides methods that return documents that should be further processed by a user in order to make use of the knowledge expressed in these documents. In the field of IE, the scale is usually smaller and the focus is much narrower. For example, in biology, IE methods generally look to extract particular types of information from texts, such as gene and protein names, their functions and interactions between proteins described in texts. It is often the case that IE techniques work on the level of a single document to extract the facts expressed in the text. These facts can be represented in a structured form, which opens up the way to use them in data mining systems to extract new knowledge or to use this data as input for further human analysis.

In the first part of this thesis we focus on the IE aspect of the text mining process applied to biology, and more in particular on extracting protein-protein interactions (PPI). Indeed, with specialized IR systems such as PubMed [126] it is easy to find thousands of texts related to a particular topic in biology, however the processing of these results still poses a problem. In the last decade a whole new subfield of text mining in biomedicine has emerged, consolidating researchers from the domains of computational linguistics and machine learning. This new area is a source of many new approaches that address the IE and data mining tasks outlined in this section.

This chapter introduces the basic concepts from machine learning and natural language processing and set the scene for machine learning applications in IE for biology. In Chapter 3, we discuss methods proposed to extract PPI descriptions from text. These methods rely on different lexical and syntactic features extracted from text. An interesting question to ask is which kind of features is more important for PPI extraction. In Chapter 3 we investigate the impact of lexical, shallow and deep syntactic features on the performance of PPI extraction methods.

The remainder of this chapter is structured as follows. In Section 2.2 we first recall the principles of natural language processing on the lexical and syntactic levels and in Section 2.3 we discuss the corpora that we will use for the interaction extraction task. In Section 2.4 we recall the formal definition of machine learning and explain the Support Vector Machines classifier we will use as well as discuss the performance metrics that can be applied to evaluate the PPI extraction task. Finally, we conclude in Section 2.5.

## 2.2 Natural language processing

Analyzing natural language is a difficult task sometimes even for human experts, let alone automated methods. The same information in natural language can be represented in an extremely rich variety of forms, and the first step to take in order to extract this information is to split the monolithic text of a document into smaller pieces that are interconnected in meaningful ways. The structured representation of a text is essential in order to apply any automated method, because such methods generally cannot work on unstructured data. The preprocessing step shapes a text structure by splitting it into tokens, recognizing specific token types and eventually by finding syntactic and semantic dependencies between tokens. Generally, natural language processing (NLP) can be divided in three stages: lexical, syntactic and semantic. The output of all three stages can be used for information extraction purposes, however currently nearly no systems are available which can accurately model the semantics of biomedical texts, thus in this section we discuss only the former two stages.

### 2.2.1 Lexical processing

Lexical processing includes two connected steps: tokenization and stemming.

#### 2.2.1.1 Tokenization

Although seemingly simple, tokenization is an important step in text processing. It is the very first step, and errors made here will propagate to other levels, deteriorating the performance of high-level tasks such as Named Entity Recognition (NER) or PPI extraction.

Tokenization is the task of segmenting the input character stream into linguistically plausible units called tokens [5]. The tokens can be classified into several categories such as words, punctuation, acronyms, numbers, Greek letters, etc. A straightforward way to perform tokenization is to separate tokens on white spaces and punctuation marks, however this may lead to a number of problems that arise from the fact that we want tokens to be *sensible*.

For example, hyphenation may be ambiguous. Consider the string *p53-dependent*, which semantically should be treated as two tokens, while *co-transcribed* apparently needs to be a single word. Moreover, cases such as *alpha- and beta-catenin* that occur in biological texts also require attention, as this string describes two entities *alpha-catenin* and *beta-catenin*. Information about entities

that occur in a text may be crucial for further information extraction tasks, thus it is important to properly handle such cases.

Another case where whitespace and punctuation tokenization may fail is the representation of number and dates. Numbers can be represented in various formats: digits can be grouped by white spaces, commas or dots or not grouped at all, and the decimal separator can vary as well, e.g. *1 123,56* may be equivalent to *1,123.56* or *1123,56*. Dates arguably should be treated as single tokens as well, but the richness of representations makes date parsing a non-trivial task too.

Moreover, some biology-specific constructs may challenge even very sophisticated tokenizers by constructs like *(26.0 +/- 1.8 vs. 19.1 +/- 2.2 micrograms/mg wet wt)* or *erythro-9-(2-hydroxy-3-nonyl)adenine*. Inline citations, that sometimes occur in biological texts are another source of potential problems, e.g. *(Crute, J.J., Tsurumi, T., Zhu, L., Weller, S.K., Olivo, P.D., Challberg, M.D., Mocarski, E.S. and Lehman, I.R. (1989) Proc. Natl. Acad. Sci. U.S.A. 86, 2186-2189)*.

Typically, the tokenization problem is handled with lexicon-based approaches and/or regular expression matching. For example, in a work of Grefenstette and Tapanainen [68] regular expressions are used to extract dates and numbers and a lexicon- and rule-based system to detect abbreviations. However, such approaches do not scale well, as the context of periods in the biological domain is very different from the one in general text.

One of the most important problems in tokenization is to detect sentence boundaries, or in other words to disambiguate periods. Abbreviations pose a problem to this task, as periods may denote abbreviations, such as in the bacteria name *E.coli* or in *U.S.* However if the latter appears at the end of the sentence, the period will denote the end of sentence as well, and it is not trivial to classify this period as an end-of-sentence token. Simple dictionary lookup may not catch all cases, since new abbreviations are constantly generated.

Alternatively, this problem can be approached as a classification task. Machine learning-based approaches provide more flexibility than rule-based ones, as a tokenization strategy is learned from the text itself, and does not contain hand-crafted rules. Palmer and Hearst [120] propose using a neural network that considers parts-of-speech tags around the period to classify it as the end of a sentence or as an abbreviation punctuation mark. Their system called SATZ achieves 98.5% accuracy on the *Wall Street Journal* (WSJ) corpus, which is a typical benchmark corpus for NLP tasks. Another approach described by Reynar and Ratnaparkhi [139] uses the tokens around a potential end of sentence period as a set of features and applies a Maximum-Entropy classifier to recog-

nize whether the period is at the end of a sentence. This approach yields a performance of 98.8% on WSJ.

#### 2.2.1.2 Stemming

Morphological analysis is used to convert various syntactic forms of a word to its canonical form. This is arguably an important step because it reduces the variation among tokens and thus reduces the sparseness in the lexical representation of the text. For example, the phrases ‘*a inhibits b*’ and ‘*a inhibiting b*’ will be cast to a single bag-of-words representation  $\{a, b, inhibit\}$  which potentially simplifies the task of recognizing an interaction between *a* and *b*. The most common algorithm for English words base form derivation was described already in 1980 by Porter [123]. The Porter stemming algorithm uses a list of English inflections such as *es*, *ed*, *ing*, *ly* etc. and strips the longest subsequence from every token considered. This algorithm, although being extremely simple, is a standard approach in tasks where a bag-of-words document representation is used.

### 2.2.2 Syntactic processing

Syntactic processing is an essential part of most NLP applications, as syntactic information sometimes exposes a connection between the structure and semantics in the text (such as the fact that a protein name most probably will be a noun phrase). Syntactic processing may be divided into three stages: part-of-speech (POS) tagging, chunking and deep syntactic processing.

POS tagging is the first step and its outcome is essential for the following processing. POS tags contain information derived from the text that can further be used in IE tasks. Usually, it is not possible to use a lexicon-based approach, i.e., to assign a POS tag looking at a word in isolation, and the surrounding context is often needed to determine it properly. For example, the word *report* can be either a noun or a verb depending on the context.

Automated approaches for building POS tagging systems can be divided into rule-induction and statistical ones. Both approaches require training data, which are texts with manually assigned POS tags. The rule-induction taggers derive the set of rules that should be used to assign POS tags to words. An example of a rule-induction tagger is Brill’s tagger [17] that takes an initial set of rules and applies it to data. A rule can make an error in POS tag assignment, which can be fixed by adjusting the rule. The algorithm extends and modifies the set of rules and iteratively estimates the error-level and finally comes



up with a ranked list of rules that provide the best POS tag assignment. This system obtains an accuracy of 96.6% on the WSJ corpus.

Many approaches to POS tagging, such as TnT [15] or HunPos [71] are based on Hidden Markov Models (HMM). An HMM contains a hidden state sequence and a known observation sequence. In a hidden state sequence every next state depends only on the previous state and does not depend on the states that follow, i.e., it is a Markovian sequence. In the case of POS tagging this is the sequence of POS tags in a sentence. An observation sequence contains the observations that are associated with hidden states, which are represented as tokens in the case of POS tagging. The parameters of the HMM model are conditional probabilities that are estimated from the training data. In the TnT tagger a dynamic programming algorithm (the Viterbi algorithm [170]) is used to find the most probable sequence of states in the model.

More sophisticated statistical POS taggers employ different approaches such as maximum entropy [162] and SVM [65]. In both of these approaches a rich feature vector is built from the context surrounding the word of interest. Toutanova et al. [162] derive from the data a (cyclic) dependency network consisting of words and POS tags with directed arcs representing the influence of the nodes on each other. Each node's neighborhood is considered in isolation, and therefore presents a non-cyclic network for which conditional probabilities can be induced from the data. At application time the sequence that maximizes the product of these local conditional probabilities is derived and used as the most probable sequence of POS tags. In this way, when predicting a POS tag for a word not only preceding words and POS tags are considered, as in traditional HMMs, but also the words from the right-hand side together with their possible POS tag assignments that are yet to be determined. This approach yields 97.24% accuracy on the WSJ corpus. The SVM approach [65] employs a very rich feature vector that considers features up to trigrams (consequent word triples) and uses a polynomial kernel SVM to obtain an accuracy of 97.2% on the WSJ corpus (see Section 2.4.2 for more details about SVM). Adapting POS taggers to the biomedical domain requires re-training on a collection of biomedical documents with the *de facto* standard choice being the GENIA [119] corpus. The performance of state of the art classifiers such as [164] that is built on the same ideas as [162] achieves an accuracy of 98.26%.

Chunking is the process of combining continuous pieces of text into phrasal sequences such as nouns, verbs, prepositions or adjective phrases, using both lexical and POS information. A more general task than chunking is full parsing, also referred to as deep syntactic processing. Full parsers try to build a complete sentence structure also known as a parse tree. Parsing is the culmination of the

NLP task as it combines the data obtained in all previous steps including POS tags and phrases and adds even more information about the structural dependencies between phrases. All modern NLP parsers are built as supervised systems, which means that the training data should contain manually constructed parse trees for every sentence in a corpus, called a treebank. A parse tree is the complete syntactic representation of a sentence, however building such representation is a costly process, therefore it is still not clear whether the additional information provided by parse trees is worth the computational efforts. This problem, in an application to the PPI extraction problem, is discussed in depth in Chapter 3.

Parser development is an active area of research that has resulted in many publicly available tools. Parsers can be divided into two categories: constituent parsers, such as the Charniak-Lease parser [92], Enju [144], Link Parser [69] and Stanford Parser [88], and dependency parsers such as MST [101] and MALT [117]. Constituent parsers produce parse trees, which contain syntactic tags as inner nodes and the actual words of a sentence as leaves. Dependency parsers produce dependency trees, which reflect the interconnection between words in a sentence, i.e., all nodes of a dependency tree are represented by words. Note, that while there is a common scheme for building parse trees, there exist several independent dependency schemes. This is explained by the fact that the notion of dependencies between words is open for interpretation (e.g. some parsers may provide untyped dependencies, while others may provide dependencies typed with syntactic or semantic roles). Some dependency schemes, such as the Stanford Dependency scheme that was developed with the application to the IE task in mind, are better suited for the task of interaction extraction than others [130]. Importantly, a parse tree structure can be converted to a dependency representation (or even to different dependency representations), while the reverse conversion is more problematic, i.e. constituent parsers can be used as the replacement for dependency parsers. Dependency parsers are much faster than constituent parsers [21], however, the latter provide better accuracy when used to build dependency trees, therefore in our work we opt for a constituent parser. A comparative study of full parsers' accuracy for biomedical text parsing is provided by Clegg and Shepherd [28].

## 2.3 Biological corpora

Machine learning approaches as well as most of the NLP components require annotated training data for learning. Another observation that we made in

Section 2.2.1, is that the language of biological texts is very different from the general domain, which implies that specific linguistic resources are needed in order to perform NLP tasks with reasonable accuracy.

GENIA [119] is one of the largest publicly available annotated corpora of biomedical texts, and contains both syntactic and semantic annotations and a treebank for parser training. This corpus is often used to retrain general-purpose NLP instruments such as parsers, named entity recognizers and tokenizers for application in the biomedical domain.

Moreover, there exists a number of smaller data sets created for different purposes. In the overview in this section we provide details for the data sets we use in Chapter 3 that are built specifically for the PPI extraction task. These publicly available data sets are AIMed [18], BioInfer [131], HPRD50 [58], LLL [114] and IEPA [41]. These data sets have been frequently used in recent work [1, 87, 66, 169]; therefore we use them in our current work. Table 2.1 gives an overview of the annotated interaction statistics in the different data sets.

The AIMed data set consists of 225 abstracts extracted from the Database of Interaction Proteins [145], 200 of which contain annotated human gene and protein interactions. These 200 abstracts contain 1955 sentences that we have used for the evaluation purposes.

The BioInfer data set is the largest data set among these 5; it contains 1100 sentences describing protein-protein interactions. Besides the interaction annotations, BioInfer contains additional information about the biological interaction type, protein roles in the interaction, syntactic dependencies between words, etc. Moreover, there is a knowledge base behind the corpus, which allows a researcher to analyze it in more detail (see [131]).

HPRD50 contains sentences that were extracted from a subset of 50 abstracts, referenced by the Human Protein Reference Database (HPRD) [86] and annotated with protein names and interactions between them.

The LLL data set consists of 77 sentences describing interactions concerning *Bacillus subtilis* transcription. Protein roles for interactions are annotated along with the interactions themselves. Additionally, the data set contains annotations for lemmas and syntactic dependencies between the words in the sentences.

Finally, the IEPA data set was built by querying MEDLINE with 10 diverse queries, reflecting 10 different biological topics. 303 abstracts were retrieved, and a data set was constructed with sentences extracted from these abstracts. The data set annotation includes an interacting verb along with the protein names and interactions.

The BioInfer and LLL data sets provide syntactic dependencies for every sentence in their own formats, while the other data sets do not provide this infor-

mation.

All data sets use different annotation schemes that emphasize different interaction properties. For example, in AIMed homodimeric proteins, i.e. proteins that interact with themselves, are annotated which may limit the utility of PPI extraction approaches that expect two entities to be involved in an interaction. Moreover, in BioInfer some proteins have gaps in annotations, i.e., there is a gap between two parts of one protein name, which can potentially decrease the performance of PPI methods as well. The quality of the annotation itself (measured as e.g. inter-annotator agreement) may affect the quality of a classifier trained on the annotated data. If an annotator misses an interaction between two proteins, the data point related to this protein pair would be treated as a negative instance, although containing an interaction pattern, which is harmful to the overall performance.

An extensive comparative analysis of the 5 datasets is presented by Pyysalo et al. [128]. One of the most important observations is that the choice of corpus has a determining effect on the evaluation figures of a PPI extraction method, thus it is important to use several corpora ('hard' ones as well as 'easy' ones) to get a better estimate of the method's accuracy. The authors hypothesize that the difference between results is correlated with the number of non-interacting entities annotated in the corpus, that can be expressed as the proportion  $I/EP$ , where  $I$  is the average number of interacting entity pairs per sentence and  $EP$  is the average number of annotated entity pairs per sentence.

To unify the experimental setup, we need to cast all corpora to a common ground format. Pyysalo et al. [128] designed custom software that converts all 5 data sets to a single XML format that contains only minimal protein and interaction annotations, which is sufficient for our training and evaluation purposes in Chapter 3. However, not all annotation differences can be eliminated in this way. Table 2.1 shows that different data sets have very different ratios of interactions per sentence. This can be partially explained by different annotation strategies, e.g. for LLL only proteins that are involved in interactions are annotated, while for other data sets all protein names are annotated. Since we consider every possible protein pair within a sentence to be an instance, this leads to a combinatorial growth of the total number of instances, while in fact the number of positive instances remains the same.

Data set	Number of sentences	Average number per sentence	
		proteins	interactions
AIMed	1955	2.2	0.5
BioInfer	1100	4.2	1.3
HPRD50	145	2.8	1.1
LLL	77	3.1	0.7
IEPA	486	2.3	2.1

Table 2.1: Corpora statistics.

## 2.4 Machine learning

Information extraction and text mining methods rely heavily on machine learning. Machine learning is generally associated with the question of how to construct computer programs that automatically improve with experience [108].

### 2.4.1 Learning problems

One frequent situation that machine learning aims to handle is when we do not have an explicit definition of how to solve the problem, but (many) example solutions are available. This setting corresponds to the supervised machine learning paradigm and captures the problem of information extraction that we tackle in this thesis. More formally:

**Definition 2.1** (Learning problem [108]). *A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .*

According to this definition, to formally define a learning problem we need to define a task  $T$ , experience  $E$  and a performance measure  $P$ . Given the information extraction problem, and more in particular the PPI extraction task we can define the learning problem as follows:

- Task  $T$ : extract the interactions between genes and proteins from English texts
- Performance measure  $P$ : f-score with respect to positive instances
- Experience  $E$ : English texts with annotated gene and protein interactions

The goal of the PPI extraction task is to learn the mapping

$$f : X \rightarrow \{0, 1\}$$

where  $f$  is called a *target function* that maps the representation of the text (which, in fact, could be a feature vector, phrase, sentence or an abstract)  $x \in X$  to 1 if the text described by  $x$  contains an interaction, and 0 if the text described by  $x$  does not contain an interaction. In reality we use a limited amount of training data that is not enough to find  $f$ , thus instead of learning  $f$  we learn the approximation of this function denoted as  $\hat{f}$

$$\hat{f} : X \rightarrow \{0, 1\}$$

which is approximated using the available training data  $E$ . For every PPI extraction method the choice of the appropriate text representation is crucial for the method's performance. In the following discussion we represent the training experience  $\mathbf{E}$  as a collection of examples  $(x_1, c_1), \dots, (x_n, c_n)$ , where  $x_i$  is a text we want to analyze, and  $c_i \in \{0, 1\}$  is a label, associated with every text.

This high-level definition covers all machine learning-based approaches to PPI extraction such as [1, 19, 66, 58, 87] to name a few. We have already described the training experience represented by annotated corpora in Section 2.3. In the next subsection we provide a description of the Support Vector Machines approach that we employ in Chapter 3 to study the impact of various textual features on PPI extraction accuracy. Further, in subsection 2.4.3 we discuss performance measures that can be used to evaluate the PPI extraction task.

## 2.4.2 Support vector machines

A machine learning algorithm that has become very popular for information extraction tasks (see e.g. [66, 87, 96]) is Support Vector Machines (SVM) [30]. To describe SVM, let us first describe the idea of an optimal hyperplane classifier that is a linear version of SVM. Consider a data set consisting of instances  $(x_1, c_1), \dots, (x_n, c_n)$  with  $c_i \in \{-1, 1\}$ ,  $i = 1 \dots, n$  and its vectorial representation  $(\mathbf{x}_1, c_1), \dots, (\mathbf{x}_n, c_n)$ . Vector  $\mathbf{x}_i$  defines the instance  $x_i$  in some feature space  $\mathcal{X}$ . Assume that these instances can be linearly separated, e.g. in the two dimensional case we can draw a line that will separate the two sets of points, belonging to different classes. A hyperplane  $H_{\mathbf{w}, b}$  is defined by the equation

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \tag{2.1}$$

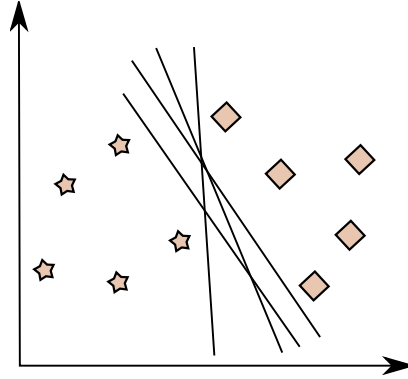


Figure 2.4.1: There can be infinitely many hyperplanes that separate the class of instances represented by stars from the class of instances represented by diamonds.

where  $\langle \cdot, \cdot \rangle$  represents the inner product of two vectors,  $\mathbf{w} \in \mathcal{X}$  is a vector orthogonal to the hyperplane,  $b \in \mathbb{R}$  is the distance from the hyperplane to the point of origin.

Next, let us define the distance function  $d$  as follows

$$d(\mathbf{x}, H_{\mathbf{w},b}) = \frac{\langle \mathbf{w}, \mathbf{x} \rangle + b}{\|\mathbf{w}\|}.$$

The hyperplane  $H_{\mathbf{w},b}$  is called a separating hyperplane if for every training instance  $\mathbf{x}_i$  it holds that  $d(\mathbf{x}_i, H_{\mathbf{w},b}) > 0$  if  $c_i = 1$ , and  $d(\mathbf{x}_i, H_{\mathbf{w},b}) < 0$  if  $c_i = -1$ . The decision function that can be used to classify a new instance  $\mathbf{x}$  is then defined as

$$\hat{f}(\mathbf{x}) = \text{sgn}(d(\mathbf{x}, H_{\mathbf{w},b}))$$

or, since  $\|\mathbf{w}\| > 0$ ,

$$\hat{f}(\mathbf{x}, H_{\mathbf{w},b}) = \text{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b). \quad (2.2)$$

As illustrated in Figure 2.4.1 there may be many hyperplanes that separate the training data. We want to pick one that generalizes best over unseen data, i.e., we would like to find a hyperplane that maximizes the ‘margin’ between the training instances and the hyperplane itself. To this end we need to solve the

following optimization problem

$$\max_{\mathbf{x}, b} C \quad (2.3)$$

$$C \leq d_{abs}(\mathbf{x}_i, H_{\mathbf{w}, b}), i = 1, \dots, n \quad (2.4)$$

where

$$d_{abs}(\mathbf{x}_i, H_{\mathbf{w}, b}) = c_i \frac{\langle \mathbf{w}, \mathbf{x}_i \rangle + b}{\|\mathbf{w}\|}$$

Note, that there can be many solutions to the problem described by (2.3) and (2.4). Indeed, we can rescale an optimal solution  $(\mathbf{w}, b)$  with a constant  $\lambda > 0$ , and it will remain an optimal solution. To eliminate the scaling problem let us fix the length of  $\mathbf{w}$  as follows

$$\|\mathbf{w}\| = \frac{1}{C}$$

Now we can rewrite the problem (2.3)-(2.4) as follows

$$\min_{\mathbf{x}, b} \|\mathbf{w}\|$$

$$c_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, i = 1, \dots, n$$

This problem can be reformulated as follows

$$\min_{\mathbf{x}, b} \frac{\|\mathbf{w}\|}{2} \quad (2.5)$$

$$c_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, i = 1, \dots, n \quad (2.6)$$

This is a constraint optimization problem that can be solved by introducing *Lagrange multipliers*  $\alpha_1, \dots, \alpha_n$ ,  $\alpha_i \geq 0$  and a *Lagrangian*

$$L(\mathbf{w}, b, \alpha) = \frac{\|\mathbf{w}\|}{2} - \sum_{i=1}^n \alpha_i (c_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1). \quad (2.7)$$

where  $\alpha$  is a shorthand for  $\alpha_1, \dots, \alpha_n$ . To solve the optimization problem (2.5)-(2.6), the Lagrangian (2.7) should be maximized with respect to  $\alpha$  and minimized with respect to  $\mathbf{w}$  and  $b$  [91]. Consequently, the derivatives of  $L$  with respect to  $\mathbf{w}$  and  $b$  must turn to zero

$$\begin{aligned} \frac{\partial}{\partial b} L(\mathbf{w}, b, \alpha) &= 0 \\ \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \alpha) &= 0 \end{aligned}$$



which leads to

$$\sum_{i=1}^n \alpha_i c_i = 0 \quad (2.8)$$

$$\mathbf{w} = \sum_{i=1}^n \alpha_i c_i \mathbf{x}_i \quad (2.9)$$

According to the Karush-Kuhn-Tucker (KKT) theorem [91] for every  $\mathbf{x}_i$  should hold

$$\alpha_i (c_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1) = 0$$

This means that either  $\alpha_i = 0$  or  $c_i (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$ . From (2.9) we can see that if  $\alpha_i = 0$ , then instance  $\mathbf{x}_i$  has no influence on  $\mathbf{w}$ . The instances for which  $\alpha_i > 0$  are used to calculate  $\mathbf{w}$  and called *support vectors*.

To find the values of  $\alpha_i$  we substitute (2.8) and (2.9) in (2.7) and arrive at the dual optimization problem that does not contain  $\mathbf{w}$  and  $b$

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j c_i c_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \quad (2.10)$$

$$\text{subject to } \alpha_i \geq 0 \text{ for all } i = 1, \dots, n \text{ and } \sum_{i=1}^n \alpha_i c_i = 0 \quad (2.11)$$

which is a quadratic optimization problem that has an optimal solution. Several techniques have been proposed to solve this problem in the context of SVM such as [48, 83].

Finally, after substitution of (2.9) into (2.2) the decision function will be of the form

$$\hat{f}(\mathbf{x}) = \text{sgn} \left( \sum_{i=1}^n \alpha_i c_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b \right) \quad (2.12)$$

Equations (2.10) and (2.12) provide the basis for the training and classification procedure for the optimal margin classifier. However, often the classes in a data set are not linearly separable. The idea of SVM is that the training data, defined in space  $\mathcal{X}$ , can be mapped into a higher-dimensional feature space  $\mathcal{H}$  through a non-linear mapping  $\phi$ , and in this feature space the classes are separable and an optimal hyperplane classifier can be applied. Consider an instance  $x \in \mathcal{X}$  from a training data set, its mapping to  $\mathcal{H}$  will be

$$\mathbf{x} := \phi(x)$$

and the inner product between instances  $x$  and  $x'$  is denoted as

$$k(x, x') = \langle \mathbf{x}, \mathbf{x}' \rangle$$

Function  $k$  is called a kernel on  $\mathcal{X} \times \mathcal{X}$ . Note that the feature vectors in (2.10) and (2.12) appear only in inner products, thus these equations can be reformulated in terms of kernels, which is called the kernel trick. It turns out that for certain feature mappings  $\phi$  that comply with Mercer's theorem [104], the kernel function can be computed in terms of input vectors without explicit transformation to a higher-dimensional space, which allows for an efficient classification algorithm. Moreover, the kernel implementation is decoupled from the classification algorithm itself and to consider another feature space we can simply change the kernel.

**Example 1** Consider two instances  $x = (x_1, x_2)$  and  $x' = (x'_1, x'_2)$  described in a two-dimensional space  $\mathbb{R}^2$ , and consider a mapping  $\phi$  that maps a vector to a higher-dimensional space  $\mathcal{H}$  that contains 2nd order products of its features (monomials), i.e.,

$$\phi(x) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$$

Here we consider ordered monomials, i.e.  $x_1x_2$  and  $x_2x_1$  are considered as different features. It is easy to write this mapping for two-dimensional vectors and for 2nd order products, however, in the general case for  $n$ -dimensional vector with  $d$ th order products the size of  $\mathcal{H}$  is

$$\binom{d+n-1}{d} = \frac{(d+n-1)!}{d!(n-1)!}$$

It turns out that we can compute the inner product of two vectors represented in  $\mathcal{H}$  in the original space. Consider the vectors  $x$  and  $x'$ , then

$$\begin{aligned} \langle \phi(x), \phi(x') \rangle &= \langle (x_1^2, x_2^2, x_1x_2, x_2x_1), (x_1'^2, x_2'^2, x_1'x_2', x_2'x_1') \rangle \\ &= x_1^2x_1'^2 + x_2^2x_2'^2 + 2x_1x_2x_1'x_2' \\ &= \langle x, x' \rangle^2 \end{aligned}$$

Thus, to find the inner product of the vectors that contain 2nd order monomials we need to square the inner product of the vectors in the initial space. This result was generalized to  $d$ th order monomials and called a polynomial kernel [12]

$$k(x, x') = \langle x, x' \rangle^d$$

To build a proper kernel, a number of non-trivial requirements need to be met [147]. Therefore, it is common to use previously defined kernels that have proved to be efficient in many situations. Examples of such kernels are the polynomial kernel

$$k(x, x') = \langle x, x' \rangle^d,$$

the gaussian kernel

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right), \sigma > 0$$

and the sigmoid kernel

$$k(x, x') = \tanh(\kappa \langle x, x' \rangle + \vartheta), \kappa > 0, \vartheta < 0$$

Finally, let us recall two propositions that will give us the basis to introduce the kernels for structured objects.

**Proposition 2.1** (Tensor products [147]). *If  $k_1$  and  $k_2$  are kernels defined respectively on  $\mathcal{X}_1 \times \mathcal{X}_1$  and  $\mathcal{X}_2 \times \mathcal{X}_2$ , then their tensor product,*

$$(k_1 \otimes k_2)(x_1, x_2, x'_1, x'_2) = k_1(x_1, x'_1)k_2(x_2, x'_2),$$

*is a kernel on  $(\mathcal{X}_1 \times \mathcal{X}_1) \times (\mathcal{X}_2 \times \mathcal{X}_2)$ . Here,  $x_1, x'_1 \in \mathcal{X}_1$ ,  $x_2, x'_2 \in \mathcal{X}_2$ .*

**Proposition 2.2** (Direct sums [73]). *If  $k_1$  and  $k_2$  are kernels defined respectively on  $\mathcal{X}_1 \times \mathcal{X}_1$  and  $\mathcal{X}_2 \times \mathcal{X}_2$ , then their direct sum,*

$$(k_1 \oplus k_2)(x_1, x_2, x'_1, x'_2) = k_1(x_1, x'_1) + k_2(x_2, x'_2),$$

*is a kernel on  $(\mathcal{X}_1 \times \mathcal{X}_1) \times (\mathcal{X}_2 \times \mathcal{X}_2)$ . Here,  $x_1, x'_1 \in \mathcal{X}_1$ ,  $x_2, x'_2 \in \mathcal{X}_2$ .*

The notion of tensor product and of direct sum is useful if we have a heterogeneous input, and different parts of this input call for different kernels to be applied. This approach was further investigated by Haussler [73] to define a kernel between structured objects as a combination of kernels that assess the respective parts of these objects. Consider the instance  $x \in \mathcal{X}$  that is composed of parts  $x_d \in \mathcal{X}_d$  with  $d = 1, \dots, D$ . Note, that according to Propositions 2.1 and 2.2 feature spaces  $\mathcal{X}_d$ 's need not be the same. Let us denote the relation

$R(x_1, \dots, x_D, x)$  that defines the set of allowed decomposition of instance  $x$  into  $x_1, \dots, x_D$ , then we can define the *R-convolution* as

$$(k_1 \star \dots \star k_D)(x, x') = \sum_R \prod_{d=1}^D k_d(x_d, x'_d).$$

where  $k_d$  is a kernel defined on  $\mathcal{X}_d \times \mathcal{X}_d$  for  $d = 1, \dots, D$ . Intuitively, we consider all possible decompositions of object  $x$  into  $x_1, \dots, x_D$  and of object  $x'$  into  $x'_1, \dots, x'_D$  and for every such decomposition compute a tensor product of kernels on corresponding parts. The direct sum over all such kernels is a final kernel that can be used for classification purposes. We further use this version of kernel computation in Chapter 3 to compute kernels on dependency trees.

### 2.4.3 Performance metrics

In principle, classifiers can be built to deal with any number of classes, however for the purpose of PPI extraction we consider only two classes: positive and negative. An accuracy metric that is often used to evaluate the performance of classification methods, computes the fraction of correctly classified instances in a given test set. However, in our case we are interested in correctly classifying positive instances, and generally do not care about the negative instances, which limits the applicability of accuracy metrics. In the PPI extraction task there are typically much more negative instances than positive ones, thus a classifier may achieve a high accuracy score due to the correct classification of negative instances alone.

The problem described above occurs in the area of information retrieval (IR) as well, therefore we adopt the evaluation metrics of recall and precision introduced in IR to evaluate the performance of PPI methods. Additionally, we employ ROC (receiver operating characteristic) curves for the analysis of a classifier's performance. Let us reproduce their definitions. Let  $TP$  denote the number of true positives, i.e., the number of positive instances that are classified as such, let  $FP$  denote the number of false positives, i.e., the number of negative instances that are incorrectly classified as positive, and analogously, let  $TN$  and  $FN$  stand for the number of true negatives and false negatives respectively. The following metrics can then be defined:

$$\begin{aligned} \text{recall} &= TP / (TP + FN) \\ \text{precision} &= TP / (TP + FP) \\ \text{true positive rate} &= \text{recall} \\ \text{false positive rate} &= FP / (FP + TN) \end{aligned}$$

Recall stands for the fraction of correctly classified instances ( $TP$ ) among all positive instances ( $TP + FN$ ) in a data set, while precision denotes the fraction of correctly classified instances ( $TP$ ) among all instances that are classified as positive ( $TP + FP$ ). Recall is sometimes called true positive rate, while false positive rate counts how many of the negative instances were wrongly classified as positive. A combined measure that takes into account both recall and precision is called F-score and defined as:

$$F = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$$

Often, a classifier's output can be ordered, i.e., the classifier also provides a degree of confidence for each prediction it makes. In this case, we can trade precision for a higher recall by lowering the confidence threshold to capture more positive instances. In this way we can build a recall-precision curve that shows the relationship between these two metrics. The closer to the top-right corner a curve is, the less precision is lost with recall growth and the better the performance of the classifier is.

Precision, recall and F-score are de-facto standards for interaction extraction evaluation. However, these metrics are very sensitive to data set skewedness, i.e., the large difference between the number of positive and negative instances. As shown in Table 2.1, this difference varies greatly for different corpora. On the other hand, ROC curves are being used in the machine learning community to evaluate classifier performance and they do not depend on data set skewedness. As shown by Davis and Goadrich [37], there exists a strong relationship between ROC and recall-precision curves, however, ROC curves have complementary features discussed below that make them worth using for PPI extraction evaluation.

The false positive rate together with the true positive rate correspond to a point in ROC space. By varying the trade-off between these two metrics we obtain a curve in ROC space. The AUC-score is the area under this ROC-curve. It can be interpreted as the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

However, this metric should be used carefully, because it is suitable to evaluate the recall of a classifier (percentage of extracted positive instances), but it gives no information about precision, and thus makes the evaluation of a classifier difficult. For example, if we increase the number of negatives 10 times, then the number of FP on average increases 10 times as well. This will lead to a significant drop in precision and consequently in F-score, but it does not influence the false positive rate:

$$\begin{aligned}\text{precision} &= TP / (TP + 10 \cdot FP) \\ \text{false positive rate} &= 10 \cdot FP / 10 \cdot (FP + TN)\end{aligned}$$

Based on this observation, we can outline an application area for both evaluation metrics. The ROC curve and the corresponding AUC value should be used to compare the performance of a classifier on different corpora, since they show the relative number of extracted positive instances. Recall-precision and F-score can be used to compare the quality of several classifiers on the same data set, since they indicate how ‘clean’ the classification is without regarding the proportion of negative instances.

## 2.5 Conclusions

The introduction of new experimental methods and adoption of knowledge dissemination techniques in the domain of biology in the last decade have led to the overwhelming growth of information, both explicitly presented in databases and hidden in papers. As the amount of published articles in biological domain keeps growing, it is important to develop the methods for automated information extraction. The task of protein-protein interaction extraction from biological texts is an important problem that attracts a lot of attention from computational linguistics and computer science communities.

In this chapter we have described the steps for building PPI extraction methods. Natural language processing is a necessary step to convert an unstructured text into a structured representation that can be used for information extraction tasks. We separate two NLP stages: lexical and syntactic processing. Lexical processing includes tokenization and stemming steps that separate texts on basic blocks and reduce the representation variance. Syntactic processing includes POS tagging and deep parsing, that is used to discover the interconnections between words in the text. Every step in this sequence produces features that can potentially be used for information extraction tasks, such as protein-protein interaction extraction. The amount of features may be overwhelming (consider, for example, a number of words that can appear in a text), therefore it is important to use the features that are potentially useful for the PPI extraction task. This is an interesting problem, that we address in Chapter 3. As our study in Chapter 3 rely on the SVM algorithm, we have provided a description of this algorithm.

Moreover, we describe five benchmark datasets that are used to evaluate PPI extraction tasks. We discuss the differences in these datasets and the potential

---

influence of these differences on a classifier performance. Finally, we recall the definitions of performance metrics, namely of recall-precision and ROC curves. We propose to use both these metrics to evaluate the performance of a classifier, as they provide complementary information. More in detail, recall-precision can be used to compare the quality of several classifiers on the same data set, while ROC curve should be used to compare the performance of one classifier on different corpora.

# 3 Feature analysis for protein interaction extraction from text

---

## 3.1 Introduction

As discussed in Chapter 2, the need for automated information extraction methods in biomedicine has become critical, and many efforts are invested in creating such methods. Recently proposed approaches for protein-protein interaction (PPI) extraction are based not only on explicit textual information that contain in publications, but also on a comprehensive language analysis that includes part-of-speech (POS) tags and deep syntactic structure extraction. To achieve state of the art performance, researchers employ lexical information (words) along with shallow syntactic information (POS) and/or deep syntactic features (grammatical structures) (e.g., [1, 19, 58, 66, 84, 87, 142, 169, 171, 172]).

As a consequence, extraction methods tend to become more complex, use more features and require more memory and computational efforts. However, little attention has been devoted to studying the individual impact of different feature types. We believe that this question is of great importance, because

1. When two types of features have a substitute rather than a complementary effect, one of them can be dropped to obtain a computationally more efficient method, and
2. Dropping one type of features might make the mining algorithm more robust

The latter reason is especially relevant for lexical features since lexicons tend to be subdomain-specific. This problem can be alleviated by combining different biological phenomena in one corpus; however in practice corpora are often built for a particular organism or a particular set of proteins. Despite this fact, it is



common practice to train and evaluate systems on the same data set with an  $n$ -fold cross-validation technique, thus partially avoiding this lexicon-dissimilarity problem which is inherent to real-life problems.

In this chapter, we study the impact of different feature types on the performance of a relation extraction system that uses a SVM classifier with kernels as its core, since at present this is the most popular choice in the relation extraction field. In particular, we use the approach suggested by Kim et al. [87], which relies on lexical, shallow and deep syntactic features represented as parts of a dependency tree, and consequently apply Occam's razor principle by cutting off the former two to get rid of all lexical and shallow syntactic information. In other words, we would like to exploit different aspects of the dependency tree and compare the net advantage that is obtained by these feature types.

To the best of our knowledge, besides us, only [142, 160, 169, 171] have looked at the impact of syntactic information in addition to lexical features for the protein interaction extraction task (all in the context of SVMs). Shallow syntactic features such as POS added to a lexical feature set are reported not to increase the performance of the classifier in the study of Xiao et al. [171], while the deep+shallow syntactic- and lexical-feature based classifier [142] showed a poor performance when the set of lexical features is limited. However, neither of these studies has looked into how much performance can be obtained by using *only* deep syntactic features. The closest to our work is [169] which compares the performance of an interaction extraction system using only lexical features versus using syntactic (both shallow and deep) features. The work presented by Tikk et al. [160], that studies different kernel methods for PPI extraction can be seen as an indirect expansion of our work. Here, the authors provide an extensive comparative evaluation of nine kernel-based methods in order to find the best performing one, but do not focus on studying the contribution of the different feature types. We highlight the difference with our work at the end of Section 3.4.

The contribution of this chapter is twofold. First, we perform an extensive evaluation of a recently published SVM-based approach [87], which was evaluated only on the LLL data set before. We evaluate it on 5 data sets (AIMed [18], BioInfer [131], HPRD50 [58], LLL [114] and IEPA [41]) that were discussed in Chapter 2 using cross-validation as well as 10 cross-data set experiments. Secondly, we compare this approach with stripped down versions which take into account different feature subsets, and we demonstrate that omitting the lexical and part of the syntactic features does not significantly change the performance of the relation extraction method.

In the remainder of this chapter, we first formalize the protein interaction

extraction problem as a classification task in Section 3.2, in which sentences containing protein pairs are represented by dependency trees. In Section 3.3, we present the various classifiers that we use in this chapter, all of them modifications of [87], and in Section 3.4 we clarify the relationship with related methodologies. We continue with a description of our experimental setup and present the results on the different data sets in Section 3.5. Our final conclusions are presented in Section 3.6.

## 3.2 Problem statement

Whereas the general PPI extraction task is concerned with finding all interactions among proteins in a given text, several assumptions are usually made to simplify it. The first assumption is that the extraction task is restricted to binary interactions, i.e., exactly two proteins are involved in the interaction. The second assumption is that the interaction is assumed to be fully expressed in one sentence, i.e., interactions which are described across several sentences are not considered. The final assumption is that the interaction extraction task is evaluated separately from the protein name recognition task. Named entity recognition (NER) is another area of text mining, which is usually performed and evaluated separately, thus it is generally assumed that interaction extraction is performed on a text with annotated protein names (e.g., [19, 50, 58, 84]). We further refer to this problem as the protein-protein interaction detection (PPID) subtask of the PPI task.

Let us consider the following sentence containing 4 protein names.

**Example 1** “In the **shaA**<sub>1</sub> mutant, **sigma(H)**<sub>2</sub>-dependent expression of **spo0A**<sub>3</sub> and **spoVG**<sub>4</sub> at an early stage of sporulation was sensitive to external NaCl.”

This sentence contains 6 protein pairs: *shaA-sigma(H)*, *shaA-spo0A*, *shaA-spoVG*, *sigma(H)-spo0A*, *sigma(H)-spoVG*, and *spo0A-spoVG*. A protein pair is a positive instance if the original sentence expresses an interaction between members of this pair, and a negative instance if they just co-occur in the sentence. In the example above, there are two positive instances, *sigma(H)-spo0A* and *sigma(H)-spoVG* while the other 4 instances are negative. As such, the task of protein interaction extraction can be treated as a classification problem, to be solved by learning a suitable decision function that can distinguish between the positive and the negative instances.

In particular, we need to choose a formal protein pair representation and

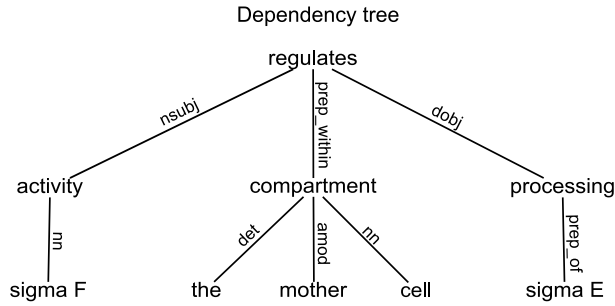


Figure 3.2.1: Dependency tree of the sentence from Example 2.

a machine learning algorithm. The protein pair representation should include information from the sentence that can be used to distinguish between positive and negative instances. The type of information (features) used, depends on the machine learning methods employed, the available tools and the researcher's strategy. Although feature selection for interaction extraction has received little attention [143], several researchers [47, 169] report that applying feature selection techniques significantly speeds up the processing and in some cases increases the performance of the classifier. The difference between the feature selection problem and the focus of our study in this chapter is explained in detail in Section 3.4.

Let us now turn to the representation of a potential interaction. We use a dependency tree because in this representation it is easy to include and exclude different types of features. A dependency tree represents the syntactic structure of a sentence. The nodes of the tree are the words of the sentence, and the edges represent dependencies between words. In a typed dependency tree, edges are labeled with syntactic roles. The dependency tree for the following sentence is depicted in Figure 3.2.1.

**Example 2** “**Sigma F<sub>1</sub>** activity regulates the processing of **sigma E<sub>2</sub>** within the mother cell compartment.”

The most relevant part of the dependency tree to collect information about the relation between the two proteins is the subtree corresponding to the shortest path between these proteins, which is shown in Figure 3.2.2a. This is a lexicalized path as it contains the words that appear in the sentence connected

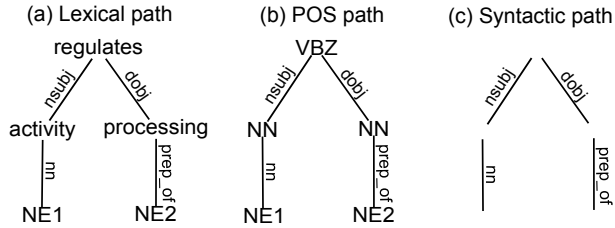


Figure 3.2.2: Lexical, POS and dependency shortest paths for the pair *Sigma F* - *Sigma E* from the sentence in Example 2.

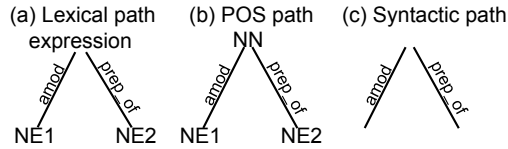


Figure 3.2.3: Lexical, POS and dependency shortest paths for the sentence in Example 1.

with syntactic relationships. From this path, we can learn that ‘NE1 activity’ is a noun phrase that is the subject of the verb ‘regulates’ and that the object of this verb is ‘processing’, which, in turn, is related to NE2 by means of the preposition ‘of’. Both protein names are replaced with dummy strings  $NE_1$  and  $NE_2$  in order to generalize the interaction pattern.

Moreover, we introduce a POS dependency tree, where nodes represent part-of-speech information instead of the corresponding words. The shortest path between the two proteins in the POS dependency tree for Example 2 is represented in Figure 3.2.2b. Note that a dependency tree contains lexical as well as deep syntactic information, while a POS dependency tree contains shallow and deep syntactic information.

We can obtain a syntactic shortest path by only retaining the syntactic roles in either the shortest path or the POS shortest path, as shown in Figure 3.2.2c. Figure 3.2.3 depicts similar information for the sentence from Example 1.

We refer to these paths as (shortest path) dependency trees. Such a dependency tree can be either a lexical dependency tree (like Figure 3.2.3a or 3.2.2a), a POS dependency tree (like Figure 3.2.3b or 3.2.2b) or a syntactic dependency

tree (like Figure 3.2.3c or 3.2.2c). We will use  $t = (N, E, L)$  to denote a dependency tree  $t$  consisting of a set of nodes  $N$ , a set of edges  $E$ , and a function  $L$  that maps nodes and edges to their labels. If there is an edge from node  $n_1$  to node  $n_2$ , we denote this edge by  $e(n_1, n_2)$ .

### 3.3 SVM with structured kernels

Our feature impact study uses support vector machines (SVM) approach which are explained in Chapter 2. SVM view input data as vectors in a high-dimensional space and attempt to induce a maximum margin hyperplane that separates training data points belonging to different classes.

In order to build a syntactic kernel, the dependency tree space has to be kernelized. Each data point of our data set is a dependency tree corresponding to a part of the sentence in which the protein pair occurs. Such a tree can be represented as a vector in the  $m$ -dimensional space made up by all subtrees in the data set [29]. In particular, assuming that all unique subtrees in the data set are enumerated from 1 to  $m$ , the function  $h_s(t)$ ,  $s \in \{1, \dots, m\}$ , is defined as the number of occurrences of subtree  $s$  in tree  $t$ . Then, each tree  $t$  can be represented by a vector  $\phi(t) = \{h_1(t), h_2(t), \dots, h_m(t)\}$ . A kernel function measuring the similarity between trees  $t_1$  and  $t_2$  based on whether they contain the same subtrees, is defined as the inner product

$$K(t_1, t_2) = \langle \phi(t_1), \phi(t_2) \rangle = \sum_s h_s(t_1) \cdot h_s(t_2) \quad (3.1)$$

The underlying vector representation is very rich since the number of subtrees of a tree grows exponentially with the number of nodes in the tree, which makes the computation of the inner product intractable. However, the right hand side of (3.1) can be interpreted as the number of common subtrees of  $t_1$  and  $t_2$ , and can be computed efficiently [29].

Kim et al. [87] follow this procedure in developing a SVM classifier based on a kernel  $K_{FULL}$  that is a combination of a kernel  $K_{LEX}$  comparing lexical dependency trees, with a kernel  $K_{POS}$  comparing POS dependency trees. Note that, because of their construction,  $K_{LEX}$  relies on lexical and deep syntactic information, while  $K_{POS}$  is based on shallow and deep syntactic features. We propose a way in which the kernel  $K_{FULL}$  can be stripped down to a kernel  $K_S$ , that uses only deep syntactic features. We compare the performance of all these kernels in Section 3.5. For ease of explanation, in this section we follow

a bottom-up approach by first defining  $K_S$ , and then extending it to the full system [87].

The trees from Figures 3.2.2a and 3.2.3a have no subtrees in common, while when we switch to a shallow or pure syntactic representation in Figures 3.2.3b,c and 3.2.2b,c, we have one common fragment, namely the subtree consisting only of the edge *prep\_of* and its adjacent nodes. In general, we use a recursive formula to compute the number of common subtrees between dependency trees  $t_1$  and  $t_2$ . This formula relies on the notion of *common child pairs* of node  $n_1$  in  $t_1$  and node  $n_2$  in  $t_2$ , i.e., the set of pairs of nodes that have parents  $n_1$  and  $n_2$  respectively, and that are connected to these parents by the same type of edge. When traversing down the trees in search of common subtrees, these are the nodes at which we want to continue our exploration.

**Definition 3.1.** Let  $t_1 = (N_1, E_1, L_1)$  and  $t_2 = (N_2, E_2, L_2)$  be dependency trees. For  $n_1 \in N_1$  and  $n_2 \in N_2$ , the set of common child pairs is defined as  $Com(n_1, n_2) = \{(x, y) | (x, y) \in N_1 \times N_2, e(n_1, x) \in E_1, e(n_2, y) \in E_2, L_1(e(n_1, x)) = L_2(e(n_2, y))\}$ .

**Definition 3.2.** Let  $t_1 = (N_1, E_1, L_1)$  and  $t_2 = (N_2, E_2, L_2)$  be dependency trees. For  $n_1 \in N_1$  and  $n_2 \in N_2$ , the number of common subtrees rooted at  $n_1$  and  $n_2$  is defined as

$$Cm(n_1, n_2) = \begin{cases} 0 & \text{if } Com(n_1, n_2) = \emptyset, \\ \prod_{(x,y) \in Com(n_1, n_2)} (Cm(x, y) + 2) - 1 & \text{otherwise.} \end{cases}$$

The recursive formula reflects the fact that a new common subtree rooted at  $n_1$  and  $n_2$  can be found either by picking 1 of the  $Cm(x, y)$  subtrees or by adding the  $x/y$  nodes, or just by staying as is (therefore +2). 1 is subtracted from the whole result to exclude the combination with the tree consisting of the  $n_1/n_2$  node only.

**Example 3** Let  $t_1$  and  $t_2$  be the dependency trees from Figure 3.2.3a and 3.2.2a respectively.  $Com(n_1, n_2)$  is the empty set for all node pairs with exception of  $Com(processing, expression) = \{(NE2, NE2)\}$ . Hence

$$Cm(processing, expression) = (Cm(NE2, NE2) + 2) - 1 = 1$$

while  $Cm(n_1, n_2) = 0$  for all other node pairs. This means that there is only one common subtree between  $t_1$  and  $t_2$ , rooted at the *processing, expression* nodes and ending at *NE2*.

Note that the calculation above of the number of common subtrees disregards node labels, i.e., it treats dependency trees as they are shown in Figure 3.2.3c and Figure 3.2.2c. Using Definition 3.2 we are now able to define a kernel  $K_S$  that looks only at deep syntactic information. It computes the similarity between syntactic dependency trees as the number of grammatical structures that they have in common.

**Definition 3.3.** *The kernel function  $K_S$  is defined as*

$$K_S(t_1, t_2) = \sum_{n_1 \in N_1, n_2 \in N_2} Com(n_1, n_2) \quad (3.2)$$

for syntactic dependency trees  $t_1 = (N_1, E_1, l_1)$  and  $t_2 = (N_2, E_2, l_2)$ .

**Example 4** Let  $t_1$  and  $t_2$  be the syntactic dependency trees from Figure 3.2.2c and Figure 3.2.3c respectively. Since  $|N_1| = 5$  and  $|N_2| = 3$ , the summation in the right hand side of (3.2) consists of 15 terms. In Example 3, we already established that all of these terms are 0 with the exception of one term that equals 1. Hence  $K_S(t_1, t_2) = 1$ .

To arrive at kernels that consider additional lexical and/or shallow syntactic information, we need to extend Definition 3.1 to look at the labels of nodes.

**Definition 3.4.** *Let  $t_1 = (N_1, E_1, L_1)$  and  $t_2 = (N_2, E_2, L_2)$  be dependency trees. For  $n_1 \in N_1$  and  $n_2 \in N_2$ , the set of common child pairs, taking into account the labels of the nodes, is defined as  $Com^{lab}(n_1, n_2) = \{(x, y) | (x, y) \in Com(n_1, n_2), L_1(n_1) = L_2(n_2), L_1(x) = L_2(y)\}$ .*

In this case and further in the section function  $L$  maps words in the tree nodes to corresponding lemmas eliminating the differences arising from different word forms. The superscript “*lab*” refers to the fact that the labels of the nodes are taken into account. The appearance of  $Com(n_1, n_2)$  in the definition of  $Com^{lab}(n_1, n_2)$  illustrates that the latter builds on the former. Furthermore, it holds that

$$Com^{lab}(n_1, n_2) \subseteq Com(n_1, n_2)$$

indicating that using syntactic trees leads to a more general approach (more nodes are explored when traversing down the trees in search for common subtrees).

The number  $Com^{lab}(n_1, n_2)$  of common subtrees rooted at  $n_1$  and  $n_2$ , can now be defined in a recursive manner entirely analogous to Definition 3.2,

however relying on  $Com^{lab}(n_1, n_2)$  instead of on  $Com(n_1, n_2)$ . Since they have different labels at the nodes, the value of  $Com^{lab}(n_1, n_2)$  might be different depending on whether a lexical dependency tree or a POS dependency tree is used. In both cases, it holds that

$$Com^{lab}(n_1, n_2) \leq Com(n_1, n_2) \quad (3.3)$$

**Example 5** Let  $t_1$  and  $t_2$  be the lexical dependency trees from Figure 3.2.2a and Figure 3.2.3a respectively. For all node pairs it holds that  $Com^{lab}(n_1, n_2) = \emptyset$  and  $Com^{lab}(n_1, n_2) = 0$ .

**Example 6** Let  $t_1$  and  $t_2$  be the POS dependency trees from Figure 3.2.2b and Figure 3.2.3b respectively. It holds that  $Com^{lab}(NN, NN) = \{(NE2, NE2)\}$  and  $Com^{lab}(NN, NN) = 1$ , while for all other node pairs  $Com^{lab}(n_1, n_2) = \emptyset$  and  $Com^{lab}(n_1, n_2) = 0$ .

The potentially different behaviour of  $Com^{lab}(n_1, n_2)$  on lexical dependency trees and POS dependency trees gives rise to the definitions of the kernel functions  $K_{LEX}$  and  $K_{POS}$  respectively. Both of them consider the tree structure when computing the similarity between trees, i.e., they both rely on deep syntactic information. In addition,  $K_{LEX}$  takes the actual words of the sentence into account (lexical information) while  $K_{POS}$  considers POS (shallow syntactic information).

**Definition 3.5** ([29]). *The kernel function  $K_{LEX}$  is defined as*

$$K_{LEX}(t_1, t_2) = \sum_{n_1 \in N_1, n_2 \in N_2} Com^{lab}(n_1, n_2) \quad (3.4)$$

for lexical dependency trees  $t_1$  and  $t_2$ .

**Definition 3.6** ([87]). *The kernel function  $K_{POS}$  is defined as*

$$K_{POS}(t_1, t_2) = \sum_{n_1 \in N_1, n_2 \in N_2} Com^{lab}(n_1, n_2)$$

for POS dependency trees  $t_1$  and  $t_2$ .

Finally, Kim et al. [87] combine  $K_{LEX}$  and  $K_{POS}$  into a kernel  $K_{FULL}$  that takes into account lexical, shallow and deep syntactic information.



Notation	Formula	Used information
$K_S(t_1, t_2)$	$\sum_{n_1 \in N_1, n_2 \in N_2} Cm(n_1, n_2)$	syntactic
$K_{LEX}(t_1, t_2)$	$\sum_{n_1 \in N_1, n_2 \in N_2} Cm^{lab}(n_1, n_2)$	lexical
$K_{POS}(t_1, t_2)$	$\sum_{n_1 \in N_1, n_2 \in N_2} Cm^{lab}(n_1, n_2)$	shallow
$K_{FULL}(t_1, t_2)$	$K_{LEX} + K_{FULL}$	lexical + shallow

Table 3.1: Summary of kernels introduced in Section 3.3.

**Definition 3.7** ([87]). *The kernel  $K_{FULL}$  is defined as*

$$K_{FULL}(t_1, t_2) = K_{LEX}(t_1, t_2) + K_{POS}(t_1^{POS}, t_2^{POS}) \quad (3.5)$$

for dependency trees  $t_1$  and  $t_2$  and their corresponding POS dependency trees  $t_1^{POS}$  and  $t_2^{POS}$ .

Notice that  $K_{LEX}$  is a refinement of  $K_S$  in the sense that all the information used by  $K_S$  is also used in the same way by  $K_{LEX}$ . As a consequence,  $K_{FULL}$  is also a refinement of  $K_S$ , enriching the deep syntactic information of  $K_S$  by lexical information (through  $K_{LEX}$ ) as well as shallow syntactic information (through  $K_{POS}$ ).

**Example 7** Let  $t_1$  and  $t_2$  be the lexical dependency trees from Figure 3.2.2a and Figure 3.2.3a respectively, and  $t_1^{POS}$  and  $t_2^{POS}$  their corresponding POS dependency trees from Figure 3.2.2b and Figure 3.2.3b respectively. One can verify that

$$K_{LEX}(t_1, t_2) = 0$$

and

$$K_{POS}(t_1^{POS}, t_2^{POS}) = 1$$

hence  $K_{FULL}(t_1, t_2) = 1$ . Notice that although the trees do not show any resemblance on the lexical level, their similarity at the more general syntactic level is picked up by  $K_{POS}$ . In Example 4, we found that their syntactic similarity is also already reflected by  $K_S$ .

A short summary of the kernels described above is provided in Table 3.1.

Method	Information	Algorithm	Data sets
[19]	lexical	SVM	AIMed
[66]	lexical shallow	SVM	AIMed LLL
[171]	lexical shallow deep	Maximum entropy	IEPA
[172]	lexical shallow deep	SVM	AIMed
[84]	lexical deep	BayesNet NaiveBayes K-nearest neighbour Ensembles	AIMed
[58]	lexical shallow deep	Hand-built rules	HPRD50 LLL
[50]	shallow deep	C4.5 BayesNet	AIMed LLL
[142]	lexical deep	SVM	AIMed
[1]	lexical shallow deep	Sparse RLS	AIMed BioInfer HPRD50 IEPA LLL
[87]	lexical shallow deep	SVM	LLL
[160]	lexical shallow deep	SVM	AIMed BioInfer HPRD50 IEPA LLL
[169]	lexical shallow deep	SVM	AIMed HPRD50 IEPA LLL

Table 3.2: Overview of features and machine learning algorithms used by PPID extraction approaches.

## 3.4 Related work

In Table 3.2, an overview of recent approaches to interaction extraction is presented along with the characteristics that are relevant in the context of our work. Below we describe these approaches in more detail.

Many approaches exploit the idea of using explicit feature vectors to represent a possible interaction. In particular, approaches based on various combinations of lexical features are very popular in the relation extraction community.

Bunescu et al. [19] propose to use the sentence context, obtained by splitting a sentence into three parts, i.e. before the first protein, between the two proteins, and after the second protein, and they combine them in predefined ways to obtain 3 types of patterns. Using this information, the authors propose a kernel that naturally emerges from the subsequence kernel described by Lodhi et al. [96] and obtain good results on the AIMed corpus. Giuliano et al. [66] start from the same pattern types, but treat them as bags-of-words, and define a global context kernel. Moreover, they define a local context kernel by taking a window of predefined size around the candidate proteins and adding more shallow linguistic information, such as the lemma of the word and some orthographic features. The resulting kernel function in this case is a linear combination of the global context kernel and the local context kernel. Their method obtains state of the art results on the AIMed and LLL data sets.

Some researchers focus on sentence structure, i.e., on the parse and dependency tree, to construct a feature vector. Xiao et al. [171] study the impact of features, starting with simple words and finishing with parse and dependency trees on the IEPA corpus, and they obtain a remarkable 90.9% F-score using a maximum entropy model with lexical and shallow syntactic features. Yakushiji et al. [172] suggest that full parsing information could be very useful in the biology domain because the distance between entities in a sentence can be much longer than in general-purpose domains. Therefore, they propose a method that builds complex predicate-argument structures (PAS), and apply an SVM with an RBF kernel to these patterns to obtain a classifier model. They evaluate this model on the AIMed data set and obtain a 57.3% F-score. Katrenko et al. [84], also focus on sentence structure and use dependency trees to extract the local contexts of the protein names, the root verbs of the sentence, and the parent of the protein nodes in the dependency tree. Classification is further done by BayesNet and ensemble classifiers. Another approach proposed by Fundel et al. [58], where a manually constructed set of rules uses information from the dependency trees and a predefined vocabulary to classify possible interaction instances. This approach is evaluated on the HPRD50 and LLL data sets, as well

as on a large-scale data set consisting of 1 million MEDLINE abstracts. The extracted set of interactions contained 40% of the HPRD interaction database.

In our own previous work [50], we proposed to abstract from lexical features and use only syntactic information to obtain a more general classifier that would be suitable for different data sets without retraining. We used features extracted from dependency and parse trees to build decision trees and BayesNet classifiers, and obtained promising results using AIMed as test data and LLL as training data.

Another group of approaches does not rely on an explicit feature vector but rather makes use of structured data as input information for the classifier. This means that structured features, such as dependency trees, can be used as an input to the classifier without any additional transformations, thus reducing the risk of losing useful information. One particular way to use structured features that we adhere to in the current chapter, is to exploit structured kernels.

The approaches [1, 87, 142, 160] are closest to our work as they also use structured kernels. Structured or convolution kernels were introduced by Hausler [73], who proposed how to compute a kernel for structured objects. This work gave rise to many tree kernel methods in the text mining domain. Although this idea is quite popular in general text mining, it has not been widely explored in the interaction extraction literature.

Saetre et al. [142] apply a structured kernel to the protein-protein interaction domain. In this approach, a mix of flat and structured features is used to calculate the similarity of two protein pairs. The flat part of the feature vector contains lexical features, while the structured part is a shortest path dependency tree, referred to as a partial tree. This definition was introduced by Moschitti [111] who studied different tree partitioning strategies and their impact on tree kernels for dependency and parse trees. Using these features, Saetre et al. obtain promising results on the AIMed data set, especially in combination with a rich lexical feature set.

In another very recent approach [1], the authors propose to use the whole dependency tree to build a classifier. They use a graph kernel that takes into account all paths in dependency trees, and exploit it with an RLS (regularized least squares) machine learning method. The experimental evaluation is performed on 5 data sets: AIMed, BioInfer, HPRD50, IEPA and LLL, and for all of them, the method shows remarkably good results.

Tikk et al. [160] propose the  $k$ -band shortest path spectrum kernel (kBSPS), that uses the concept of  $v$ -walks in dependency trees, which is a sequence of nodes of length  $l$  connected with labeled edges. kBSPS is computed on the shortest path dependency trees but also has the option to extend this path with

all nodes that lie within  $k$  steps in the original dependency tree. To accommodate the rich lexical variety of dependency tree nodes, the tolerance degree for computing the similarity of v-walks is introduced. The tolerance degree varies for nodes, candidate interaction agents and edge labels and can be adjusted with the appropriate parameters. The kernel was evaluated on 5 data sets: AIMed, BioInfer, HPRD50, IEPA and LLL, and has shown the best results for HPRD50 and IEPA. Moreover, the authors provide a comparison of nine kernels on the five above mentioned data sets. These kernels used shallow linguistic features, parse and (lexicalized) dependency trees and combinations thereof. Their conclusion was that the parse trees are less useful for PPI extraction, while there is no clear leader among the kernels that use shallow features and dependency trees.

Collins and Duffy [29] developed a tree kernel that counts the number of common subtrees, but used it for parsing and not for interaction extraction. Kim et al. [87] apply this kernel to a dependency tree and to a modified dependency tree with POS instead of words, and propose a combined kernel, which is a sum of these two. This approach obtains state of the art performance on the LLL data set. In the same paper they describe a flat feature vector-based approach that also utilizes dependency trees to extract graph walks as features. In their work [169], the authors study the relative feature importance for the latter approach by using the gain ratio feature selection technique. Moreover, they study the impact of different feature types as well by comparing the performance of methods that use syntactic features versus methods that use lexical features. Our approach is also based on Kim's work, however it is different from the aforementioned in several aspects. First of all, our approach is different from the feature selection task, because we focus on the *type* of the information (lexical, POS, grammatical relations) rather than on separate features. In other words, we do not use feature selection techniques to discriminate useful individual features, but fit an existing relation extraction method to consider only a subset of features of a certain type, and study the impact of this feature class. Secondly, when we study the impact of different feature types we do not rely on a flat vector, but on a structured representation. Moreover, we use an additional data set and define more extensive experimental setups in order to perform a complete study of different use cases.

A recent interesting work of studying feature importance for the biological event extraction problem (which is different from the PPID task, but shares the same principles) is presented by Van Landeghem et al. [168]. Here, the authors build a feature vector by combining words, v-walks of dependency trees, certain trigrams and manually constructed keyword features and consecutively apply

an ensemble feature selection method to reduce the number of features that are actually used for classification. The authors observe that syntactic features are highly ranked by the feature selection method, while lexical features and trigrams carry less event-related information.

## 3.5 Results

### 3.5.1 Experimental setup

In our experiments used the adaptation of SVMLight [83] which is able to handle tree-based input [111]. We build the 4 SVM classifiers that use the  $K_{FULL}$  kernel,  $K_{POS}$  kernel,  $K_S$  kernel and  $K_{LEX}$  kernel. Furthermore, we organized 3 experimental setups. The first setup uses 10-fold cross-validation (CV), where each data set is split into 10 parts, of which 9 are used for training and one for testing. Despite the fact that this is the most common way of evaluation, it should be used carefully. Since we work on instance level, it can be the case that two nearly identical instances from the same sentence fall into a train and a test fold at the same time. This ‘leak’ can cause a performance boost [142, 169].

In the second setup (4-1) we join 4 data sets to form a training set, and use the remaining one as a test set. Compared to CV, this alternative experimental setup is closer to a real world situation where information for processing is obtained from different sources and the lexicon is not as uniform as in one precompiled data set.

In the third setup (1-4), we use 1 data set as training set and the remaining 4 as test sets, thus making another step to the real world. Typically, biologists have a very limited amount of annotated data compared to the size of available unlabeled information. We try to model this situation by making the training set much smaller than the test set.

For each experimental setup we run all classifiers with all data set combinations. An analysis of the results obtained is provided in the following section.

### 3.5.2 Discussion

Table 3.3 and Figures 3.5.1 - 3.5.4 give an overview of the evaluation results for all experimental setups. In line with our evaluation metric review in Section 2.4.3, we use recall-precision and ROC curves to analyze the obtained experimental results. On the basis of these results, we can make the interesting observation that the performance of different kernels is roughly comparable, while

---

the amount of information they use is very different. In the analysis below we will omit the +syntactic postfix when talking about the lexical+syntactic ( $K_{LEX}$ ), shallow+syntactic ( $K_{POS}$ ) and lexical+shallow+syntactic ( $K_{FULL}$ ) kernels.

Data set	Exp.	Synt kernel		Sh+Synt kernel		Lex+Synt kernel		Lex+Sh+Synt kernel	
		F-score	AUC	F-score	AUC	F-score	AUC	F-score	AUC
AIMed	CV	0.33	0.69	0.37	0.66	0.37	0.67	0.39	0.7
	4-1	0.33	0.67	0.35	0.66	0.35	0.68	0.4	0.72
	1-4	0.23	0.64	0.24	0.63	0.22	0.61	0.24	0.67
BioInfer	CV	0.3	0.69	0.29	0.68	0.29	0.75	0.34	0.75
	4-1	0.3	0.69	0.32	0.68	0.25	0.65	0.31	0.7
	1-4	0.31	0.68	0.34	0.67	0.26	0.64	0.35	0.7
HPRD50	CV	0.58	0.69	0.59	0.73	0.44	0.72	0.56	0.73
	4-1	0.48	0.73	0.48	0.72	0.47	0.75	0.56	0.75
	1-4	0.33	0.67	0.31	0.64	0.26	0.63	0.3	0.65
IEPA	CV	0.76	0.81	0.71	0.8	0.66	0.7	0.72	0.8
	4-1	0.35	0.69	0.32	0.64	0.19	0.56	0.29	0.68
	1-4	0.36	0.66	0.33	0.63	0.3	0.55	0.33	0.62
LLL	CV	0.74	0.81	0.74	0.76	0.67	0.67	0.76	0.73
	4-1	0.43	0.68	0.44	0.71	0.31	0.6	0.39	0.74
	1-4	0.37	0.67	0.34	0.63	0.3	0.55	0.33	0.62

Table 3.3: Evaluation results for the experiments on 5 PPI datasets.



Data set	CV	4-1	1-4	Most useful types
AIMed	Synt	Lex Lex+Sh	Lex	Lex
BioInfer	Lex	Lex	Lex+Sh	Lex
HPRD50	Lex	Lex Lex+Sh	Lex	Lex
IEPA	Lex+Sh	Synt Lex+Synt	Synt	Biased to Synt
LLL	Synt	Lex+Sh	Synt	Synt
Result	Not clear	Lex+Sh	Not clear	

Table 3.4: Summary of the information types that provided the best results in different experimental settings.

Precision plays a particularly important role in the interaction extraction task, because if the extracted information is processed by a biologist, s/he would not like the system if it produces too much rubbish. Therefore, we are particularly interested in the left side of the recall-precision charts, where precision is typically high, although recall may be quite low.

Table 3.3 shows the F-measure results calculated for classifier confidence threshold 0.5, as well as the AUC values which are not dependent on any threshold. Let us note that the F-measure values could be tuned up, because on most recall-precision charts precision does not drop significantly after a certain point, while recall keeps growing. However, this is not the aim of the current research, thus we leave the task of looking for the optimal operation point aside. Table 3.4 sums up our empirical observations over recall-precision curves provided in Figures 3.5.1 - 3.5.3 particularly in the area before 0.2-0.3 recall values. For example, Figure 3.5.1d clearly shows that the lexical kernel performs best for the recall up to approximately 0.18 in this experimental setting. These empirical observations serve as the basis for building Table 3.4. The table gives some insight on how different types of information affect the performance of the relation extraction system. Note that these observations are different from Table 3.3, because typically a 0.5 threshold covers more than 30% of recall. Below we provide a more detailed analysis of the results shown in Table 3.3 and in Figures 3.5.1 - 3.5.3.

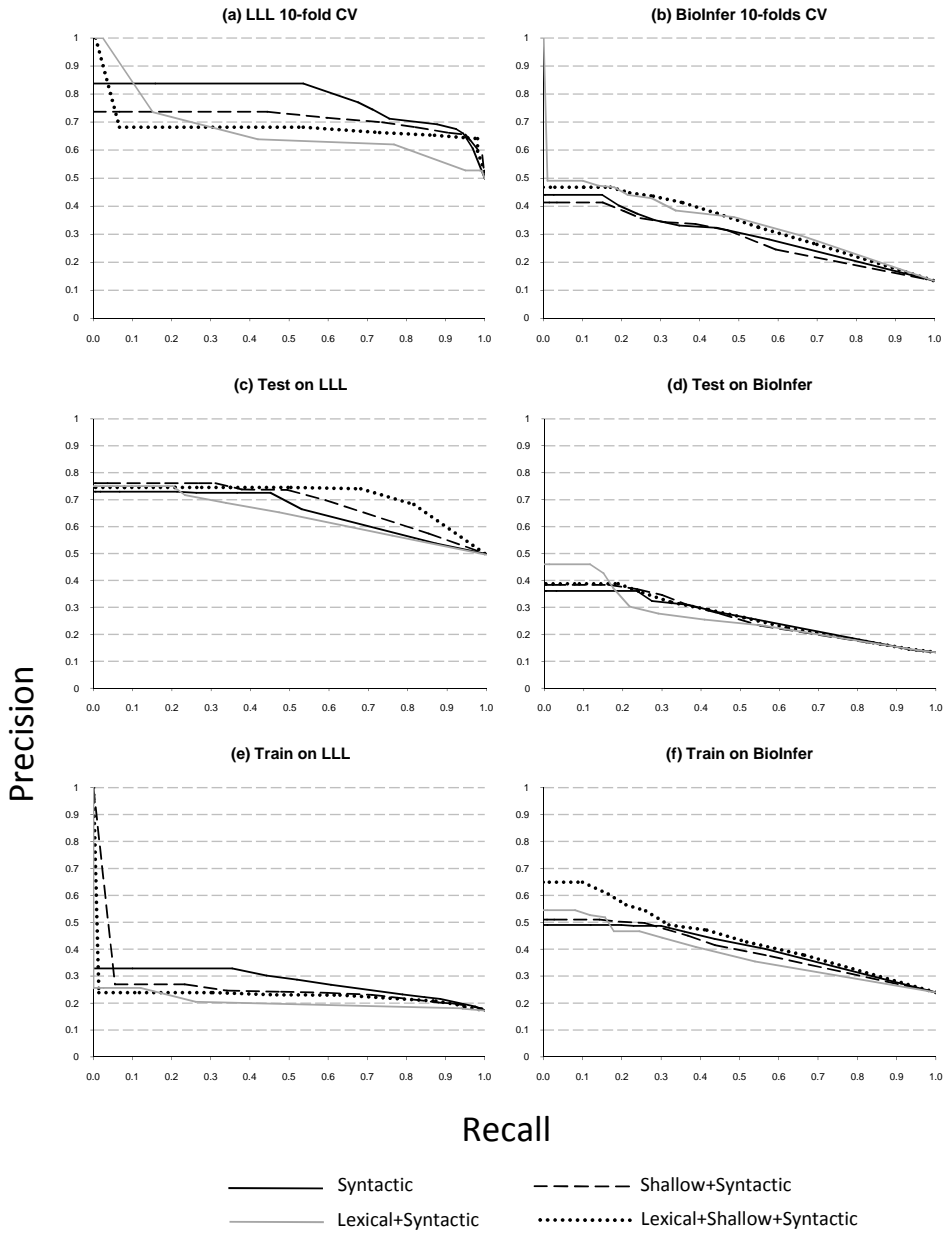


Figure 3.5.1: Recall-precision curves for all experimental setups for LLL and BioInfer.

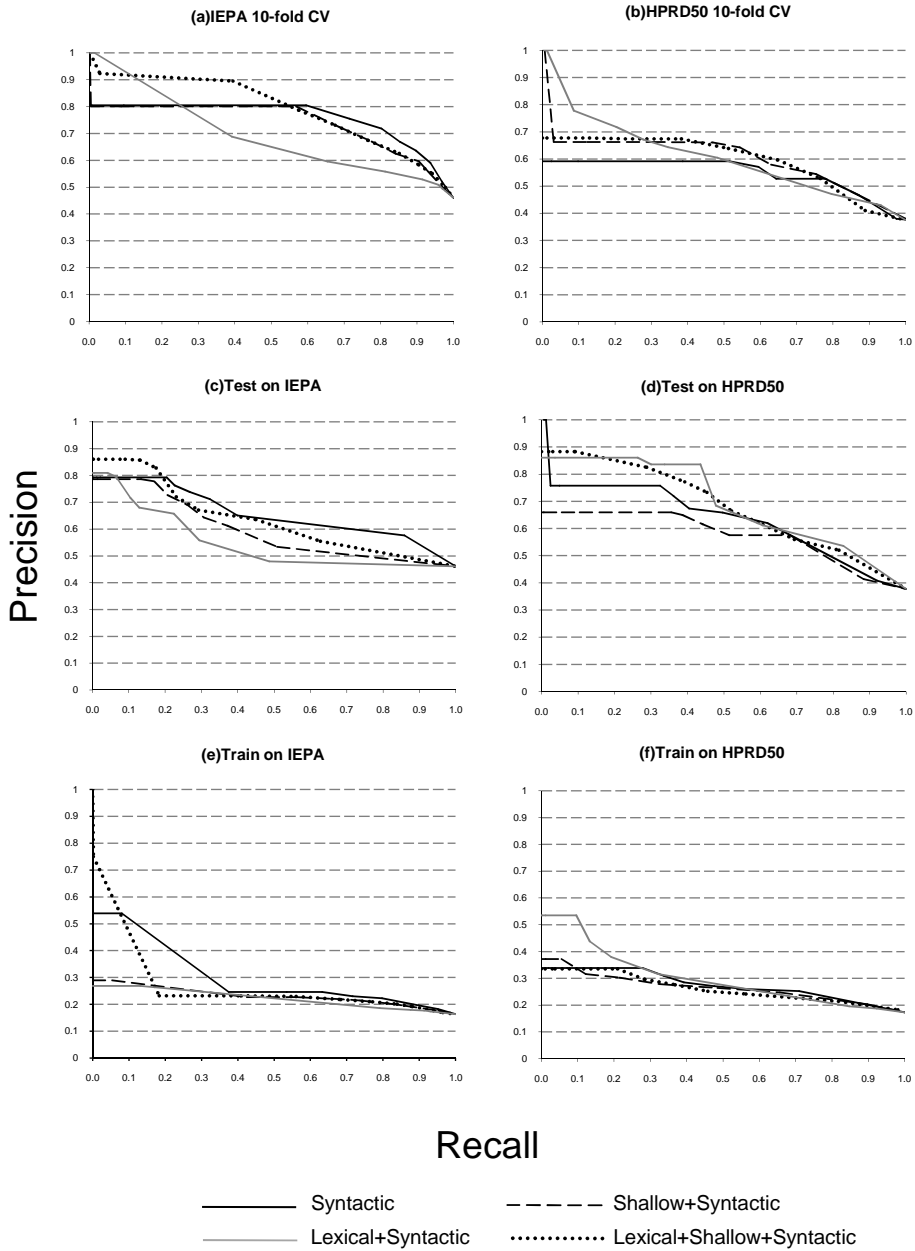


Figure 3.5.2: Recall-precision curves for all experimental setups for IEPA and HPRD50.

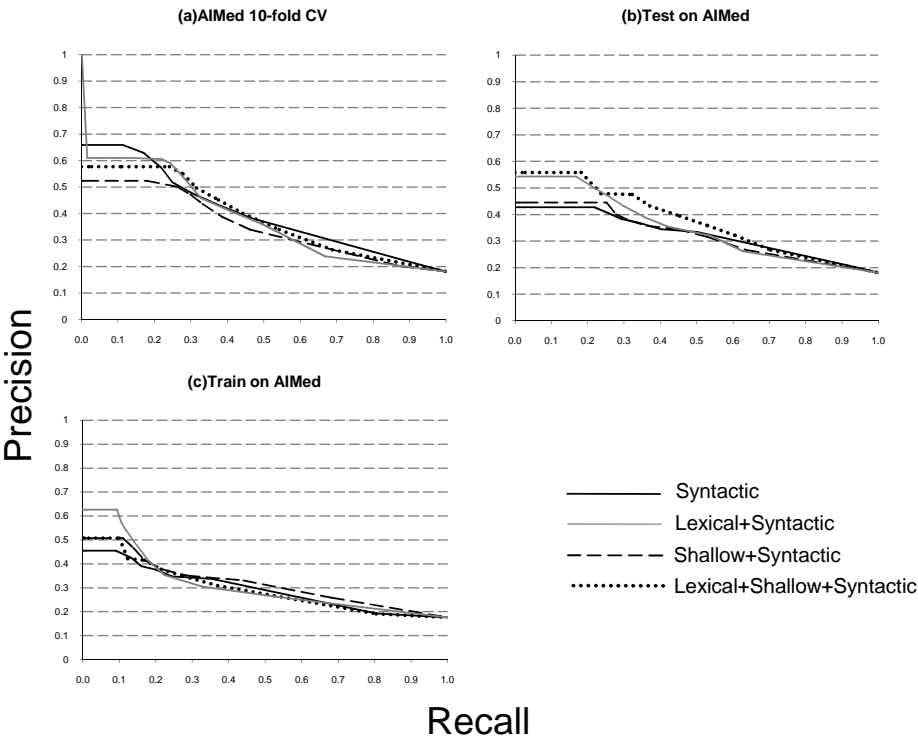


Figure 3.5.3: Recall-precision curves for all experimental setups for AIMed.

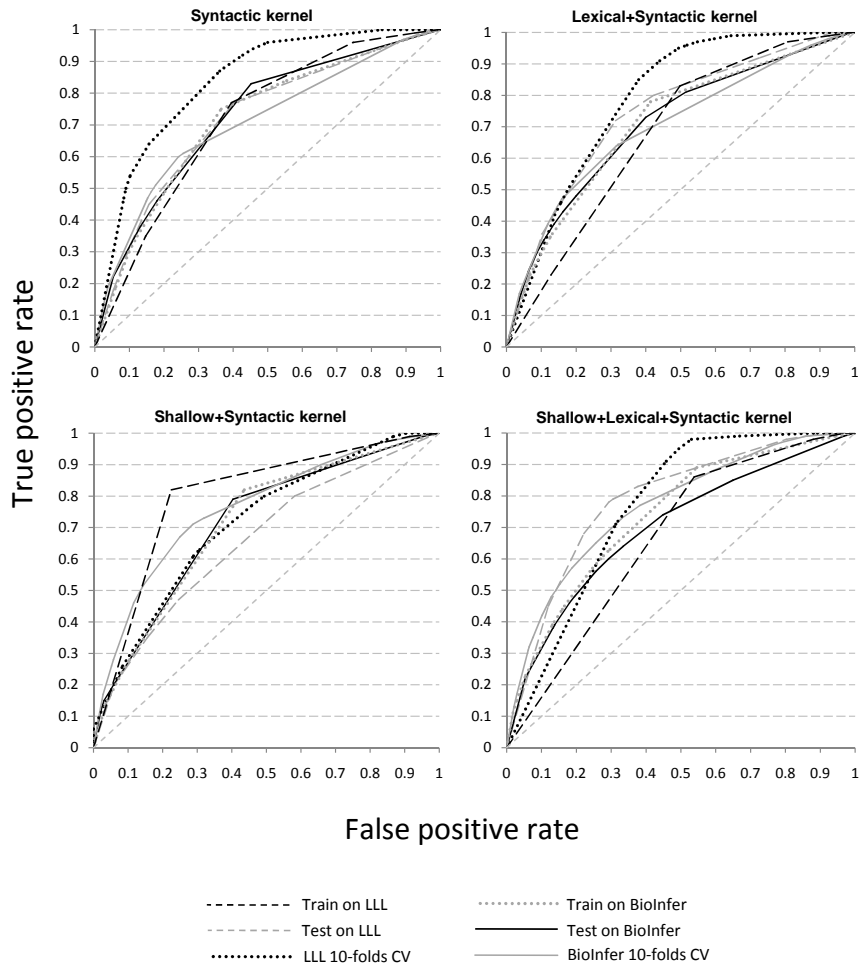


Figure 3.5.4: ROC curves for different classifiers.

The cross-validation setup reveals no clear leader for all data sets. For the LLL data set, the syntactic kernel shows the best performance (Figure 3.5.1a). This can be explained by the fact that the LLL data set is very small and contains relatively short hand-picked sentences with a simple syntactic structure. However, experiments with other data sets show that the lexical kernel gives the best results for the HPRD50 and BioInfer data sets (Figures 3.5.2b and 3.5.1b). In the case of BioInfer, this can be explained by the fact that the training set size is large enough to discriminate useful lexical features. For IEPA, the full kernel, i.e., lexical+shallow, performs best, while the lexical kernel shows the worst result (Figure 3.5.2a), and for the AIMed data set the syntactic kernel shows better results for small recall values (Figure 3.5.3a). The predictive power of deep syntactic features by themselves is very interesting, given that the lexical and lexical+shallow methods in theory can take additional advantage of the lexicon similarity within the same data set that is caused by the nature of the cross-validation set up.

When we train on 4 data sets and test on 1, the lexical+shallow kernel is among the best for all but the BioInfer data set. Figure 3.5.1d shows that the lexical kernel outperforms the others on BioInfer for small recall values. A significantly better performance of the lexical kernel for small recall values can be interpreted as a sign of overfitting, i.e., a classifier with a lexical kernel produces too specific patterns, which causes a successful classification of several instances, but is followed by a significant precision drop due to the inability to generalize over less frequent cases. On the other hand, other classifiers avoid overfitting and a steep precision drop, but at the cost of missing some very reliable patterns. Moreover, the lexical kernel shows a performance similar to the lexical+shallow kernel for HPRD50 and AIMed (Figures 3.5.2d and 3.5.3b), but fails on IEPA (Figure 3.5.2c). On the other hand, the syntactic kernel performs good on the IEPA and LLL data sets (Figures 3.5.2c and 3.5.1c), but is not that good on others.

Although with the 1-4 experimental setup there is no best kernel either, we can still observe some interesting patterns. The lexical kernel shows a significantly better performance for small recall values when trained on the AIMed and HPRD50 data set (Figures 3.5.3c and 3.5.2f), while the syntactic kernel performs best on the whole curve when trained on the LLL and IEPA data sets (Figures 3.5.1c and 3.5.2e). As shown in Figure 3.5.1e, training the classifier on LLL results in extreme curve shapes caused by the significant difference in size between the training set and the test data set. The first instances for the lexical+shallow and the shallow kernels were classified correctly, but further precision drops dramatically. After the 0.25 recall value, the lexical kernel basi-

cally neglects all positive instances, and the curve shows simply the percentage of positive instances in the data set. Other kernels perform slightly better and the syntactic kernel is able to consistently outperform others. This can be explained by the fact that 80 sentences (the size of the LLL data set) is definitely not enough to train a classifier. Moreover, it shows that in the case of training information shortage the syntactic kernel can offer a better solution than others.

The last two experiments illustrate the case when the vocabulary of train and test data sets differ, which is often the case in the real world. In the former case the training set is large enough to successfully train the lexical+shallow kernel, making the difference in the vocabularies not so crucial. However, in the latter case, when the training set is much smaller than the test set (train on LLL case on Figure 3.5.1e) we can clearly see the influence of this fact on the performance difference between syntactic and lexical methods.

From the experiments above we can observe the following trends:

- Lexical and combined methods are able to build better generalizations (due to large amount of available lexical data) and thus perform better with large (relative to test) training sets
- Syntactic methods are able to achieve better results than lexical ones when the training set is small in comparison with the test set

Moreover, there seems to be a correlation between better performing kernels and data sets. For example, the syntactic kernel always obtains good results on the LLL and IEPA data sets, while the lexical+shallow kernel performs well for the BioInfer data set. Moreover, the lexical kernel is always on top for the HPRD50 data set. These observations show that the data set origin and properties such as annotation strategy, have a strong influence on classifier performance.

Compiling ROC curves for one method on one chart allows us to analyze the robustness of this method on different data sets. In Figure 3.5.4, each chart displays ROC curves for one method for all experimental setups. The less spread the curves are in the ROC space, the more predictable the performance of the method is. In most cases, the LLL cross-validation setup is out of the trend, because of its small size and density. Otherwise, the shallow (Figure 3.5.4c) and syntactic (Figure 3.5.4a) kernels exhibit more or less coherent behaviour for all setups for the given data sets. The lexical+shallow kernel (Figure 3.5.4d) shows some spread, but again mostly due to the LLL data set's based setups, and the lexical kernel (Figure 3.5.4b) proves to be the most unpredictable.

## 3.6 Conclusion

In this chapter, we have examined different structured kernels with SVM's to study the impact of different features on the relation extraction process. We took four kernels that reflect different degrees of using syntactic and lexical information and performed three types of experiments to study the behaviour of these methods under different conditions. We performed our experiments on five benchmark data sets, being AIMed, BioInfer, IEPA, HPRD50 and LLL.

The most important observation is that by using only grammatical relations (syntactic kernel) we can obtain a similar performance as with an extended feature set (lexical kernel). This indicates the relative importance of grammatical information for the interaction extraction task. Another finding is the correlation between training/test set sizes and the method choice. We observed that when the training set is much smaller than the test set, then the syntactic kernel performs better. This might be explained by the fact that there are too few instances to induce useful lexical features, whereas syntactic features require less instances to produce better results.

When the training set grows, the performance of the full kernel becomes better, and when the training data set is larger than the test set (which rarely happens in real life), the full kernel outperforms all other kernels. From the stability point of view (i.e., the expected performance on unseen data), we can conclude that the syntactic kernel provides the best results, whereas the lexical kernel provides the worst results.

We believe that these findings can be helpful in building faster and less complicated classifiers, as well as for choosing a proper kernel according to the data set at hand.

The question of how different features within one feature type affect the quality of classification still remains open and represents an interesting direction for future work.



# 4 Systems biology and computer science

---

## 4.1 Introduction

One of the ultimate goals of biology in general is being able to understand the structure of complex living organisms, which would allow explaining an organism's development cycles and its responses to the environment. The discipline of systems biology pursues this goal by attempting to create in-silico models that reflect the behaviour of real living organisms [141]. The value of accurate models is extremely high as they, for example, can predict the influence of a drug on an organism or provide insights on how organism development could be directed without performing many costly wet lab experiments, but rather using cheap computational resources. Moreover, such models could provide feedback for future research directions as they could aid in developing new hypotheses regarding organism properties.

To build such models, systems biology adopts the *holistic* viewpoint, as opposed to the *reductionist* viewpoint widely adopted by many scientific fields including biology itself. The idea of the reductionist approach is in reducing the complexity of a research subject by dissecting it in parts. Every part then is analyzed independently to determine its function in the research subject. An illustration to this approach would be the view of cellular biology for most of the 20th century: "genes are transcribed to mRNA, mRNA is translated to proteins, proteins carry the biological functions; ergo, listing the genes allows us to understand all characteristics of the organism" [167]. However, scientists realized that proteins and genes are combined in complex networks that define their functions in an organism, thus another view on the subject was needed. The holistic approach takes an opposite position to reductionism by suggesting that

different parts of the subject are tightly connected to each other and perform their function only in connection with other components, thus the only way to study the subject is to consider it as a whole.

Following this approach, systems biology combines data from disciplines such as genomics, transcriptomics, proteomics and metabolomics (collectively known as ‘omics’ data) and tries to build models that assimilate this data using mathematical frameworks and methods. Recent advances in high-throughput experimental techniques have led to a rapid growth of experimental results, however, this data is poorly structured and error prone [148]. Many of the existing modelling formalisms are not directly applicable to such data, and this fact gave rise to new approaches aimed at accommodating the large amounts of available data to build consistent and comprehensive models.

Nevertheless, building all-subsuming models is still beyond the reach of current approaches, thus these models still focus on different representation levels. Metabolic networks modelling is concerned with modelling biochemical reactions where the concentrations of different chemical compounds change over time, thus modelling formalisms should be capable of representing consumption/production processes [14]. Signaling networks that model pathways or signal transduction systems are concerned with the events that accompany signal propagation and response to this signal, not the chemical interactions per se, thus different modelling approaches are needed in this case [154]. Gene regulatory networks are again different as genes affect each other indirectly, thus their activity is defined by interactions with proteins which in turn may call for specific modelling formalisms [36].

In this chapter we provide an overview of various modelling formalisms that take different views on the modelling problem and provide solutions to solve it. In Section 4.2 we explain several general approaches viz.  $\pi$ -calculus, Petri Nets and Pathway Logic, while in Section 4.3 we focus more specifically on another approach that is based on Boolean networks as this approach serves as a basis for our research explained in Chapter 5 and Chapter 6.

## 4.2 Modelling approaches

According to Fisher & Henzinger [57] the models built to analyze biological systems can be divided in two camps: mathematical and computational. These two types provide different insights in the model structure and functions due to the nature of the modelling language used. Mathematical models typically present the model as a set of equations. The building block of a mathematical

model is a transfer function that describes a relationship between input and output quantities. The composition of various transfer functions describes a biological model, which is analyzed by solving these equations.

In computational approaches the model is typically defined as a (possibly non-deterministic) state machine that defines how a model switches between certain states. The main difference between mathematical and computational approaches is that computational methods lend themselves well to computer simulation as the models themselves describe a set of instructions that should be executed, while mathematical models require non-trivial approximation methods in order to simulate the model on a computer.

Mathematical models are represented by various differential equation techniques (see e.g. [38]). Such models require specific mathematical skills and a lot of empirical data (kinetic data) to build, which makes their construction costly and time-consuming. Kinetic data is difficult to measure and it depends on the environmental conditions which are often not normalized in the experiments and thus it is impossible to combine it from different experiments. Note that the large amount of omics data available from high-throughput techniques is typically qualitative data that describes concentrations and expression levels, while kinetic data deals with exact quantitative values [148]. While being extremely useful in cases when such information is available, mathematical models are difficult to apply when some data is missing or when the system is iteratively expanded with new components.

Computational modelling on the other hand, provides more flexibility as some approaches make it possible to build models solely from qualitative data, while others allow for a mix of qualitative and quantitative data. Many of these approaches are rooted in computer science formalisms and were adapted to accommodate the needs of systems biology. In the remainder of this section we give an overview of the major computational methods for modelling biological systems.

### 4.2.1 $\pi$ -calculus

One large group of computational modelling approaches, many of which are described by Ciocchetta and Hillston [27], is based on process algebras [107] that were initially developed to build and analyze concurrent computer systems. The distinguishing features of process algebras are formality - the system can be described using a formal and unambiguous framework, compositionality - the system can be built by defining the interactions between explicitly defined components (compartments) and abstraction - the ability to build high-level

systems that may disregard the internal behaviour of comprised components. Regev et al. [138] developed an approach for  $\pi$ -calculus, a process algebra formalism, that defines the semantics of  $\pi$ -calculus elements in the context of biological modelling. In this approach, protein molecules and their domains are represented as processes and interactions are represented as communication channels between these processes.

Let us consider a toy example describing the *cyclin* and *cdk* binding process in the cell cycle to get a highlight of syntactic features provided by  $\pi$ -calculus. *Cyclin* activates the *cdk molecule* by binding to it, and activated *cdk* may trigger other interactions or may fall back to an inactive state if the bound *cyclin* is degraded. We start our  $\pi$ -calculus program with the following statement

$$(1) \text{ CELL\_CYCLE } ::= \text{ CYCLIN} | \text{CDK}$$

Which defines a main process *CELL\_CYCLE* and two components of this process, viz. *CYCLIN* and *CDK* which are the molecular compounds represented as processes themselves. A vertical bar means that these two processes may be executed in parallel, i.e. that both *cyclin* and *cdk* may exist and act at the same time. The following statement

$$(2) \text{ CYCLIN } ::= (\text{new } bb) \text{ BINDING\_SITE}$$

defines the behaviour of *CYCLIN*, by saying that it is defined by the behaviour of another process *BINDING\_SITE*. The expression in brackets defines the communication channel for the subprocess *BINDING\_SITE*. Further in the program *BINDING\_SITE* may use the channel *bb* to send messages to other processes. In the next statement

$$(3) \text{ BINDING\_SITE } ::= \overline{\text{bind}} < bb > . \text{CYCLIN\_BOUND}$$

the process *BINDING\_SITE* is allowed to send a message over the channel *bind*. The overline over the channel name means that the message in brackets is being sent. The content of this message is the name of another communication channel *bb* that was defined in statement (2). Another process, that listens to channel *bind* will receive the name of channel *bb* and will be able to use it for private communication with process *BINDING\_SITE*. This concept is called *name-passing* and it allows a user to dynamically define connections between processes, thus changing the network structure over time. Dot (.) defines a sequential execution of the statement, thus after sending the message *BINDING\_SITE* will behave as the *CYCLIN\_BOUND* process (not shown

in this program). The next statement

$$(4) \quad CDK ::= bind(bb).CDK\_ACTIVE$$

defines the behaviour of another component of *CELL\_CYCLE*, *CDK*. It reads the message from channel *bind* (which is the name of another channel as we defined in (3)) and then behaves as *CDK\_ACTIVE*. Essentially, rules (3) and (4) model the binding process of *cyclin* and *cdk*, where the binding is presented by the fact that *CYCLIN\_BOUND* and *CDK\_ACTIVE* now share a communication channel *bb*. The next statements

$$\begin{aligned} (5) \quad CDK\_ACTIVE &::= INACTCDH1 + NEWCDK \\ (6) \quad INACTCDH1 &::= \overline{cdh1r}.CDK\_ACTIVE \\ (7) \quad NEWCDK &::= degc.CDK \end{aligned}$$

define the further behaviour of *CDK\_ACTIVE*. It can proceed in two mutually exclusive ways, which is denoted by '+'. One option is to repress the activity of *cdh1* (the process itself is not shown) by sending an (empty) message over channel *cdh1r* and then returning to the initial state (rule (6)). Another option is that cyclin subunits degrade when a message over channel *degc* is received (rule (7)). The part that sends this message is not shown here. When the cyclin molecule degrades, the cdk molecule returns to its inactive state which is reflected in the last part of rule (7). From this state its behaviour is again defined by rule (4).

The above example provides a purely qualitative way to model protein interactions. To build more fine-grained models,  $\pi$ -calculus was extended to stochastic  $\pi$ -calculus [125] that incorporates interaction rates in the model.  $\pi$ -calculus and stochastic  $\pi$ -calculus were applied to model a number of biological networks as described by several authors [34, 93, 124].

The  $\pi$ -calculus description language provides a great toolbox to model many aspects of actual biochemical systems such as parallel execution, modularity, compositionality.  $\pi$ -calculus provides means to execute the model and observe its evolution over time, however it provides little support for reasoning otherwise, i.e. it is not clear how reachability of a state (the set of states that lead to a given state) and prediction can be checked and/or explained.

### 4.2.2 Petri Nets

Another set of approaches to model biological networks derived from the concurrent systems modelling domain is based on Petri Nets [113, 122]. An early

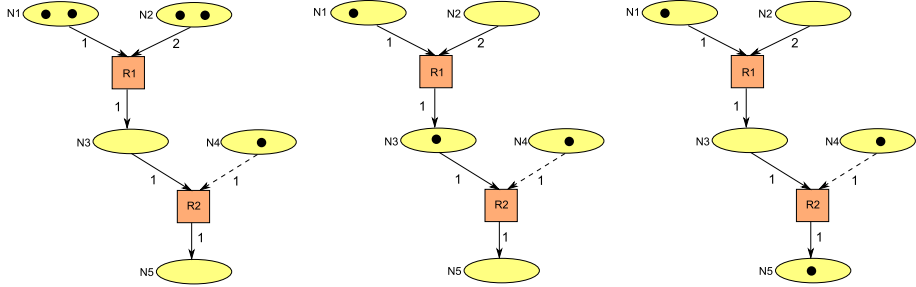


Figure 4.2.1: An example of a Petri net and its evolution over time

attempt to apply Petri Nets to model biological networks was described by Reddy et al. [137]. Here the authors defined how the elements of Petri Nets can be mapped to biological entities and processes and explored the properties of Petri Nets that are useful to analyze the behaviour of interaction networks.

Petri Nets are a formalism to model and analyze concurrent discrete event systems. Informally, a Petri net is defined as a directed bipartite graph that contains two types of nodes: *places* and *transitions*. Weighted directed edges are used to connect places to transitions and transitions to places. *Tokens* can be assigned to places, and the combination of tokens in places, called *marking*, describes the *state* of a Petri net. Every transition has a set of input and output nodes, and is said to be *enabled* if the input nodes contain a number of tokens that is greater than or equal to the weight of the corresponding edge. If a transition is enabled, it can *fire* by taking tokens from input nodes and placing them to the output nodes. The number of tokens that are taken and placed is defined by the weight of the corresponding edge.

**Example 1** Consider the network depicted in Figure 4.2.1 on the left. This is a Petri net with places  $N1, N2, N3, N4, N5$ , one enabled transition  $R1$  and one disabled transition  $R2$ . The initial marking of this network is  $[2, 2, 0, 1, 0]$ . After the transition  $R1$  is fired, the tokens were consumed from the places  $N1$  and  $N2$  and one token is placed in the place  $N3$ , enabling transition  $R2$ . This transition has an input from place  $N4$  denoted by a dashed line, which means that the transition does not remove the token from that place. After firing  $R2$  the network arrives to the rightmost state in Figure 4.2.1, where no more transitions are allowed.

Reddy et al. [137] model an oxidative pentose phosphate pathway and the main glycolytic pathway of the erythrocyte cell by representing chemical compounds as places and interactions between them as transitions in a Petri net. Tokens represent the concentration of every compound in an abstract way. The model was then analyzed using Petri Nets analysis techniques that proved to be useful for biological networks. These techniques include structural reduction, boundedness checking and liveness. Structural reduction is a method to ‘collapse’ certain parts of a Petri net into smaller parts without altering the global network behaviour. This method may be useful to abstract from certain network components, which structure is not yet well studied. Boundedness checking is another technique that does not depend on any marking. This technique analyzes whether the number of tokens in a node is bounded by a finite number, thus allowing to detect the possibility of uncontrolled accumulation of a certain compound. Liveness detects the reachability of the transitions in a network, and can be used to determine deadlocks that prevent a certain reaction from completion.

A recent overview of Petri Net-based modelling for biology have been done by Chaouiya [22]. The authors discuss different extensions of the Petri Net formalism such as Coloured Petri Nets that were used to simplify modelling of genetic networks [24] and Stochastic Petri Nets that were used to model uncertainty in biological models (see e.g., [67]). The relations between Petri Nets and *pi*-calculus were discussed by several authors [20, 40, 110] where the translation of different fragments of  $\pi$ -calculus into Petri Nets is proposed.

### 4.2.3 Pathway Logic

Another interesting approach to model biological networks was proposed by Eker et al. [45, 46] and is called Pathway Logic. A very good introduction to Pathway Logic is presented by Talcott [157]. This approach stems from rewriting logic [105] - a logical formalism designed to represent concurrent systems. In rewriting logic, states are represented as elements of certain types and the transitions between the states are represented by *rewrite rules*. A user can define his/her own types such as proteins, genes, cell locations, etc., and define elements of this type. A rewrite rule is of the form  $t \rightarrow t' \text{ if } c$  where  $t, t'$  are patterns represented as sets of elements (possibly with variables) and  $c$  is the condition for rule execution. If the condition  $c$  is satisfied, the rule can be applied, which means that the part of the system state that matches the pattern  $t$  is rewritten with the pattern  $t'$ . A system can be executed by iteratively applying

rewriting rules until the desired state is reached, or until no more rules match the current state.

Pathway Logic is implemented in the Maude<sup>1</sup> language that provides a formal specification for the state and rule description. The description of elements and rules comprise a knowledge base that captures the semantics of a certain biological system. The knowledge base description may use variables to e.g. represent a family of proteins that may interact with some specific protein, and thus it provides an abstract way of representing information. To actually reason about a biological network, a specialized tool called Pathway Logic Assistant (PLA) is used. This tool transforms the representation in a knowledge base to an intermediate knowledge base that can be transformed to Petri Nets. This representation instantiates the knowledge base variables with values, but also allows for graphical model representation. As the derived network can be very large, PLA supports queries to specify the parts of the network that are of interest for biologists. These queries can be of three types: goals, avoids and hides. Goals specify the state that should be reached in the network, avoids specify the states that should not be encountered during the network execution and hides specify the rules (reactions) that should not be fired during the network execution. Once the queries are issued, PLA computes the relevant subnetwork of the original network and model checking is employed to assess whether the goal is reachable in this subnetwork. Other properties of the derived subnetwork can be analyzed using the Petri Nets techniques described above. Pathway Logic has been used to model different biological systems, see e.g. [161, 156].

### 4.3 Boolean networks

The approaches described above provide very fine-grained model description capabilities, which can be used to build detailed and precise models and obtain informative predictions. On the other hand, the amount of available information does not always allow a biologist to build such detailed models, thus sometimes it is necessary to use coarser-grained modelling tools to analyze the system. In the following section we describe boolean networks that can be used for this purpose.

---

<sup>1</sup><http://maude.cs.uiuc.edu>



### 4.3.1 General definitions

Boolean networks are one of the first formalisms employed to study gene regulatory networks and were proposed by Kauffman [85]. A boolean network captures interactions between genes and proteins (further referred to as ‘entities’) in the form of a directed graph  $G = (V, E)$  with  $V$  a set of nodes and  $E$  a set of edges. The nodes represent entities while the edges represent the influence of one entity on another. At any given time, a node is in one of two states: either it is active, denoted by 1, or it is not active, denoted by 0 (hence, the name Boolean networks). The state of a regulatory network at any given time is defined in terms of the states of its nodes.

**Definition 4.1** (Network state). *Let  $G = (V, E)$  be a graph representing a regulatory network. Then a mapping  $S : V \rightarrow \{0, 1\}$ , that maps every entity in  $V$  to a state in  $\{0, 1\}$ , is called a network state.*

Every node may have *input nodes* that are determined by the inbound edges, and *output nodes* that are determined by the outbound edges of the node. For every node in the network a deterministic transition function can be defined that determines the next state of the node depending on the node’s inputs. The network can switch from one state to another by applying such update functions on its nodes. The transition function can be represented as a boolean function (as e.g. in Figure 4.3.1), although different representations are also possible as discussed later in this section. For the purpose of this work we consider a network transition function that is merely a combination of transition functions on the nodes. Other transition functions are possible in case of e.g. completely asynchronous network where only one node is updated at a time.

**Definition 4.2** (Transition function). *A function  $f(S) = S'$  that maps a network state  $S$  to another network state  $S'$  is called a transition function.*

To analyse the dynamics of Boolean networks, discrete time is usually considered, i.e. there is an external ‘clock’ that consequently iterates over the values 1, 2, ... At every time step, a transition function is applied to the network which causes a change of the network state at the next time step. The network evolution over time is called a trajectory and is formally described below.

**Definition 4.3** (Trajectory). *A sequence  $T$  of network states  $S_0, f(S_0), f(f(S_0)) \dots$  is called a trajectory of the network.*

For notational convenience we use  $S_i$  to denote the  $i$ th element of the trajectory, i.e. the state that is obtained after applying  $f$  to some initial state  $S_0$   $i$  times.

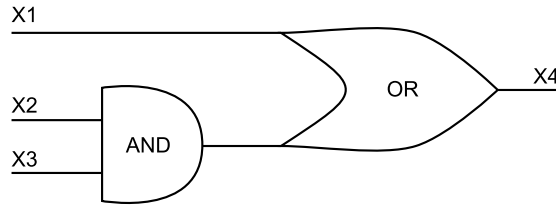


Figure 4.3.1: A transition function representation example that is equivalent to the boolean equation  $X_4 = X_1 \vee (X_2 \wedge X_3)$

Due to the deterministic nature of the network, at most after  $2^{|V|}$  steps the network will visit a previously visited state and either will stay in this state, or start to loop through a set of visited states. Such states are called *attractors* or *steady states* and are defined below.

**Definition 4.4** (Steady state, steady cycle). *A state  $S$  of a network is called a steady state if  $f(S) = S$ . A subsequence  $S_m \dots S_n (m < n)$  of a trajectory is called a steady cycle if  $f(S_n) = S_m$ . The set of trajectories that lead to a given steady state or cycle is called a basin of attraction.*

A number of studies applied this approach to model regulatory networks, for example [3, 2, 49, 70]. Moreover, translations of Boolean networks to other formalisms were studied, e.g. Maurin et al. [100] discussed the translation of Boolean networks to  $\pi$ -calculus and Chaouiya et al. [23, 24] described rewriting Boolean networks in Petri Nets.

### 4.3.2 Random Boolean Networks

In the late 1960s Kauffman proposed to study Random Boolean Networks (RBN) in order to understand the behaviour of gene regulatory networks [85]. At that time little was known about the structure of real gene regulatory networks, thus using random networks was a reasonable choice. A RBN is a boolean network that is characterized by the average connectivity  $K$  of the nodes (how many input edges has a node in the network on average) and the set of boolean functions that are allowed to determine the transition function (certain boolean functions such as tautology may be disallowed). The connections between the nodes and the boolean functions are assigned at random and the behaviour of

the system is studied through computational simulations. This analysis revealed a surprising behaviour of these networks that depends on the parameter  $K$ . With  $K \leq 2$  the network shows ordered behaviour, which means that nodes in the network change their state relatively infrequently and stable cycles are short, while when  $K > 2$  the network exhibits chaotic behaviour when nodes change their state very often, and stable cycles have a length exponential in the number of nodes in the network. This approach, although being abstract and not backed by any biological data, provided useful insights in how gene regulatory networks are organized and provided a basis for a research domain that still attracts a lot of attention (see e.g. [4], [149]).

### 4.3.3 Generalized boolean networks

Another approach that involves Boolean networks for regulatory network analysis was proposed by Thomas & Kaufman [159]. This approach extends the notion of Boolean networks to Generalized boolean networks, where a node can take more than two states to better approximate the concentrations of chemical compounds. This generalization better assimilate the observed data and consequently obtain more realistic simulation results. Moreover, the study of particular network patterns, called feedback loops, was performed [153, 159], where it was shown that the analysis of these loops helps to analyze network dynamics. This approach was applied to model regulatory networks, see for example [103, 158].

### 4.3.4 Threshold boolean networks

Threshold networks are a subset of Boolean networks [140]. In these networks the transition function on the node is defined as the sum of input signals on the node. To define these networks, let us introduce the notion of network marking.

**Definition 4.5** (Network marking). *Let  $G = (V, E)$  be a graph representing a regulatory network. Then a mapping  $M : V^2 \rightarrow \{-1, 0, 1\}$ , that maps every pair of nodes to  $\{-1, 0, 1\}$ , is called a network marking.*

Intuitively, given the pair of nodes  $\langle v_1, v_2 \rangle$ , a negative marking denotes the existence of the suppression edge between  $v_1$  and  $v_2$ , a positive marking denotes the activation edge between  $v_1$  and  $v_2$  and 0 denotes the absence of the edge. The transition function for the node can then be defined as follows

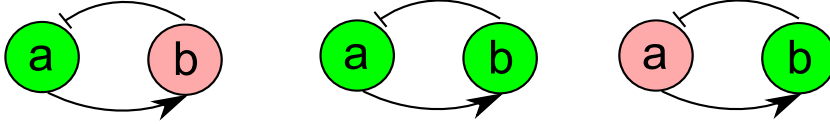


Figure 4.3.2: Threshold boolean network execution. Green node denotes active state, red node denotes inactive state

$$S_{t+1}(i) = \begin{cases} 1 & \sum_{j=1}^{|V|} M(j,i)S_t(j) + h > 0, \\ 0 & \sum_{j=1}^{|V|} M(j,i)S_t(j) + h < 0, \\ S_t(i) & \sum_{j=1}^{|V|} M(j,i)S_t(j) + h = 0, \end{cases} \quad (4.1)$$

where  $S_{t+1}(i)$ ,  $S_t(j)$  are states of node  $i$  at time point  $t + 1$ , and of node  $j$  at time point  $t$  correspondingly,  $j$  and  $i$  are the nodes of the network, and  $h$  is a threshold parameter. By setting  $h = 0$  we obtain an intuitive interpretation of a regulatory network where a gene is activated if it receives a positive input and inhibited if it receives a negative input.

**Example 2** An example of threshold boolean network and its evolution over time is depicted in Figure 4.3.2. The network contains two nodes  $a$  and  $b$ , and  $a$  activates  $b$  while  $b$  inhibits  $a$  thus forming a negative feedback loop. In its initial state  $a$  is active and  $b$  is inactive, thus according to the function (4.1) the state of  $a$  does not change and  $b$  becomes active at the next time step. In the following step,  $a$  becomes inhibited because  $b$  is active and it has an inhibition edge to  $a$ .

## 4.4 Conclusion

In the last decade biological network modelling attracted a lot of attention from the theoretical computer science field. As we discussed above, there are many attempts to adopt existing modelling frameworks to the domain of biological modelling. However, there is still no general agreement on the tools that suit the needs for any particular modelling task. Some approaches, such as Petri Nets, are best suited for modelling metabolic networks since they provide a nat-

ural representation for production/consumption processes, while others, such as Boolean networks, are more suitable for gene regulatory network modelling, since gene states are easily approximated by boolean states. For different networks a different level of details can be derived from experimental data and from the literature, which makes approaches such as Pathway Logic too complicated to study models where e.g. no compound localization is provided and the amount of information about the interactions is limited.

On the other hand, Boolean networks provide an extremely simple abstraction that allow a biologist to construct a model using a relatively small amount of information. The drawback of Boolean networks is their perturbation-observation nature, i.e. the model acts as a black box, and no means to explain the behaviour of the model itself are provided. Another problem of this approach is that the model itself may contain crucial information in an implicit way, i.e. it may be difficult to reproduce the behaviour of the model without investigating implicitly presented features such as thresholds in the case of threshold networks.

Another problem that is common for all approaches above is that it is not clear how easy it is to integrate new information in the model, and whether it can be done in a (semi) automatic way, thus leveraging the model construction with data and text mining techniques that extract gene and protein interactions from the otherwise unused data.

In the following chapter we aim to address these issues and propose a formal framework that on one hand proposes a clear and comprehensive description of Boolean networks, while on the other hand provides means for their analysis and provides a trajectory query mechanism. Moreover, the framework that we propose offers the possibility of a modular representation which means that it can be easily extended with the information that comes from external sources in a convenient way.

# 5 Modelling time-dependent systems with Answer Set Programming

---

## 5.1 Introduction

When modelling biological networks it is often desirable to have a highly configurable system where concepts such as gene, interaction, positive regulation, etc., can be defined globally in the system so that they should not be redefined for every interaction in the model. On the other hand, it is extremely difficult to describe every possible type of interaction that would suit every particular case in advance. A biological network modelling tool thus should provide enough flexibility to model new interactions while being abstract enough to allow easy manipulation of its building blocks.

Declarative languages provide a possible solution to this problem as they provide a mechanism to abstract away from the actual *procedural* implementation of the interplay between entities and interactions. These languages focus on the *declarative* level of expressing our knowledge about genes, proteins and interactions. Information about the entities and interactions between them is represented as a knowledge base, and the network modelling is performed through reasoning over this knowledge base.

Initially, declarative languages were implemented based on classical logic. However, since classical logic is monotonic, which means that conclusions obtained from a knowledge base remain to hold no matter how this knowledge base is extended later, it was not possible to model situations where conclu-

sions may change when we observe a more complete picture. This fact led to the development of nonmonotonic logics that allow overcoming this problem; an overview of different approaches have been done by Brewka [16]. One of the most fruitful achievements in the field was the establishment of *answer set* or *stable model* semantics that gave birth to a new declarative programming paradigm Answer Set Programming (ASP)[62].

ASP has already been successfully applied to different tasks in the biological domain. Ray [134] proposes an ASP-based reasoning system that explains how a particular biological model can be extended in order to assimilate a new observation. Another use of ASP was described by Gebser et al. [59] where it is used to repair biological networks when their structure does not correspond to the observed data. In this thesis, and more particularly in Chapter 6, we develop a framework for modelling biological networks in ASP. Our approach extends the boolean network modelling formalism by the possibility to model ‘meta-interactions’, i.e. the influence of entities on other interactions, not just on other entities. Moreover, we augment our framework with a flexible query mechanism that simplifies the analysis of the constructed models.

Time is an important notion in biological network modelling as it is essential to describe the rules of network evolution. For example, the rule

$$act(a, T + 1) \leftarrow act(b, T).$$

intuitively says that if  $b$  is active at time step  $T$  then  $a$  should be active at the next time step as well. A biological network can be described by a set of rules like that, and by executing these rules it can evolve from one state to another. Eventually, a network will either settle in a steady state, i.e. it will not change its state at the following time points, or it will oscillate over a set of states, i.e., it will converge to a steady cycle. Steady states and cycles of a biological network model often correspond to important states of a modelled living organism, and thus we are interested in finding these states. However, it is generally not known how many time steps it takes a network to reach a steady state, i.e., the upper time bound is unknown. Traditional ASP solving techniques cannot solve this problem efficiently, as for every new time bound the ASP program should be solved from scratch, thus repeating the computations for previously observed time states over and over again.

In this chapter, we develop efficient solving techniques that take a variable time bound into account. To this end, we introduce a specific class of ASP programs called Markovian programs. This type of time-dependent programs is defined in such a way that every next state directly depends only on the previous state, and does not depend on any of the future states (hence the name

Markovian). In Section 5.2 we recall ASP preliminaries, then in Section 5.3 we formally define time-dependent programs and in Section 5.4 we define Markovian programs, while proposing a method to solve these programs efficiently in Section 5.5. We explain the difference with other approaches in Section 5.6 and finally conclude in Section 5.7.

We intentionally keep the description of this formalism decoupled from a specific biological application to emphasize the general nature of Markovian programs. Further, in Chapter 6 we illustrate the utility of Markovian programs in biological network modelling.

## 5.2 Answer set programming

Answer set programming [62] is a declarative formalism that expresses relations between truth values of propositions with rules of the form  $\alpha \leftarrow \beta$ . Such a rule intuitively states that whenever  $\beta$  is true, proposition  $\alpha$  should be true as well. The basic building blocks of answer set programs are constants, denoted by lower-case strings (e.g.  $a, b$ ), that represent the entities; variables, denoted by upper-case strings (e.g.  $X, Y$ ) that are substituted by constants during the program grounding stage; and predicates (e.g.  $protein(a)$ ,  $activates(a, X)$ ) that represent properties of, or relations between entities.

Answer set programming allows two types of negation: classical negation denoted by  $\neg$ , and negation-as-failure (naf) denoted by *not*. Classical negation is used to express the information that we know to be negative, e.g.  $\neg act(a)$  says that  $a$  is not active. Negation-as-failure is used to express the lack of knowledge, e.g. *not*  $act(a)$  says that we have no evidence that  $a$  is active, as opposed to the classical negation.

Formally, answer set programs are built from a signature  $\sigma = \langle \gamma, v, \pi \rangle$ , where  $\gamma$  is a set of *constant symbols*,  $v$  is a set of *variable symbols*, and  $\pi = \bigcup_{i=1}^m \pi_i$  ( $m \in \mathbb{N}$ ) is the union of sets  $\pi_i$  of  $i$ -ary *predicate symbols*. To denote the arity of a predicate  $p \in \pi_i$  we use notation  $p/i$ . To facilitate the definition of time-dependent programs, in addition, we define a set of *variable expressions*  $\epsilon$  containing expressions of the form  $t' \pm t''$  where  $t' \in v$  and  $t'' \in \gamma$ . This construct is introduced to express the number addition and is handled during grounding as discussed below.

An *atom* over  $\sigma$  is an object of the form  $p(t_1, \dots, t_n)$ , where  $p \in \pi_n$  and  $t_i \in \gamma \cup v \cup \epsilon$  for each  $i \in 1 \dots n$ . A *literal* over  $\sigma$  is either an atom, or an atom preceded by  $\neg$ . *Naf-literals* over  $\sigma$ , denoting negation-as-failure, are of the form *not* $l$ , where  $l$  is a literal over  $\sigma$ . For a set of literals  $X$ , we introduce the notation



$\text{not } X = \{\text{not } l \mid l \in X\}$ . For a literal or a naf-literal  $l$ , we use  $\text{vars}(l)$  to denote the set of variables contained in  $l$ . If  $\text{vars}(l) = \emptyset$  then  $l$  is called *ground*.

In general, a rule is of the form

$$L_0 \leftarrow L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n. \quad (5.1)$$

in which  $L_0, L_1, \dots, L_n$  are literals.<sup>1</sup> If each  $L_i$  is ground, it is called a *ground rule*. Given a rule  $r$  of the form (5.1):

- The left-hand side of the rule is called the head, and defined as  $\text{head}(r) = L_0$ .
- The right-hand side of the rule is called the body, and defined as  $\text{body}_r = \{L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n\}$ .
- The set of all positive literals in the body of the rule is defined as  $\text{pos}(r) = \{L_1, \dots, L_m\}$ .
- The set of all literals in the body of the rule preceded with *not* is defined as  $\text{neg}(r) = \{L_{m+1}, \dots, L_n\}$ .
- The set of all literals is defined as  $\text{Lit}(r) = \{\text{head}(r)\} \cup \text{pos}(r) \cup \text{neg}(r)$ .

A fact is a rule with an empty body. We usually write these as ' $\alpha$ .' instead of ' $\alpha \leftarrow \cdot$ '.

**Definition 5.1** (Answer set program). *A set of rules of the form (5.1) over a signature  $\sigma$  is called an answer set program, or program for short. Similarly to the definitions on the rule level, the following shorthand notations are defined for a program  $P$ :*

$$\begin{aligned} \text{pos}(P) &= \bigcup_{r \in P} \text{pos}(r) & \text{Lit}(P) &= \bigcup_{r \in P} \text{Lit}(r) \\ \text{neg}(P) &= \bigcup_{r \in P} \text{neg}(r) & \text{head}(P) &= \bigcup_{r \in P} \{\text{head}(r)\} \end{aligned}$$

If all rules in  $P$  are ground, it is called a *ground program*. The process of *grounding* constructs a ground program  $\text{Gnd}(P)$  from an answer set program  $P$  over a signature  $\sigma$  by replacing each rule  $r$  by the set of rules obtained from  $r$  by all possible substitutions of the constants of  $\sigma$  for the variables in  $r$ . If any of the predicate arguments takes on a composite form  $t' \pm t''$  with  $t', t''$  grounded as numbers, they are substituted with the resulting value.

**Example 1** Consider the following program  $P$  with signature

<sup>1</sup>We do not consider constraints of the form  $\leftarrow \beta$  in our formal language, it can be simulated by a rule  $l \leftarrow \text{not } l, \beta$ , where  $l$  is a literal not occurring in the program.

$\langle \{a, b\}, \{Y\}, \{protein, act, activates\} \rangle$

```

protein(a).
protein(b).
act(Y)      ←  activates(X, Y), protein(X), protein(Y).

```

The grounding of this program will be as follows

```

protein(a).
protein(b).
act(a)      ←  activates(a, a), protein(a), protein(a).
act(a)      ←  activates(b, a), protein(b), protein(a).
act(b)      ←  activates(a, b), protein(a), protein(b).
act(b)      ←  activates(b, b), protein(b), protein(b).

```

Moreover, we use two extra syntactic constructs, first proposed by Simons [152]: conditional literals and cardinality constraints. Both constructs are only allowed in the body of a rule. A *conditional literal* is of the form

$$p(X_{i_1}, \dots, X_{i_n}) : p_1(X_{i_1}) : \dots : p_n(X_{i_n})$$

where  $\{i_1, \dots, i_n\} \subseteq \{1, \dots, n\}$ . In the grounding phase variables  $X_{i_1}, \dots, X_{i_n}$  are grounded with the range of values defined by the corresponding predicate  $p_i$ .

**Example 2** Consider the program

```

p(a).
p(b).
q(c).
w      ←  p(X) : q(X).

```

After grounding the program will be as follows

```

p(a).
p(b).
q(c).
w      ←  p(a), p(b).

```

A *cardinality constraint* is a construct of the form

$$B\{L_1, \dots, L_n, \text{not } L_{n+1}, \dots, \text{not } L_{n+m}\}T$$

where  $L_i$  are ground literals and  $B, T \in \mathbb{N}$  with  $B \leq T$ . A cardinality constraint holds with respect to a set of ground literals  $S$  if

$$B \leq |S \cap \{L_1, \dots, L_n\}| + (m - |S \cap \{L_{n+1}, \dots, L_{n+m}\}|) \leq T.$$

This construct was introduced by Simons [152] as an extension of ASP where the authors define a new semantics for the programs containing it. In Chapter 6 we use this construct, however, we do not consider the extended semantics as it is shown that cardinality constraints can be expressed by means of the standard ASP semantics [8], which is described below.

A rule  $r$  that does not contain negation-as-failure, i.e.  $\text{neg}(r) = \emptyset$ , is called a *simple rule*. A program that contains only simple rules is called a *simple program*.

To define what it means for a program to be solved, we recall the concepts of consistency and interpretation. A set of positive ground literals  $S$  is said to be consistent if it does not contain the literals  $a$  and  $\neg a$  together. An interpretation of a program  $P$  over a signature  $\sigma$  is any consistent set of positive ground literals  $I$  over a signature  $\sigma$ .

**Definition 5.2** (Model). *An interpretation  $I$  is a model of a ground simple rule  $r$  iff  $\text{pos}(r) \not\subseteq I \vee \text{head}(r) \in I$ . An interpretation  $I$  that is a model of all rules of a ground simple program  $P$  is called a model of  $P$ .*

**Definition 5.3** (Answer set of a simple program). *An interpretation  $I$  is called a minimal model or an answer set of a ground simple program  $P$  iff  $I$  is a model of  $P$  and there is no model  $K$  such that  $K \subset I$ .*

**Example 3** Consider the rule  $p \leftarrow q$ . The possible models of this rule are  $\{p, q\}$ ,  $\{p\}$  and  $\emptyset$ . Indeed, if  $q$  is in the model, then  $p$  should be in the model by Definition 5.2; if there is no information on  $q$  then we can make any conclusion about  $p$ , which means that  $p$  can be present or absent. The minimal model of the rule is  $\emptyset$ .

The concept of an answer set is extended for programs containing negation-as-failure as follows. Suppose that an interpretation  $I$  is a model of a program  $P$  (with negation-as-failure), and our hypothesis is that  $I$  is an answer set of

$P$ . Then we first transform  $P$  into a simple program  $P^I$  with respect to the hypothesis  $I$  and solve this program as explained above. More formally,

**Definition 5.4** (Gelfond-Lifschitz transformation). [62] *Let  $P$  be a ground answer set program. For an interpretation  $I$ , let  $P^I$  be the program (called the reduct program) obtained from  $P$  by deleting (1) all rules that contain a naf-literal  $\text{not } L$  with  $L \in I$  and (2) all the naf-literals from the bodies of the remaining rules, in other words,*

$$P^I = \{\text{head}(r) \leftarrow \text{pos}(r) \mid r \in P, I \cap \text{neg}(r) = \emptyset\}$$

**Example 4** A program with negation-as-failure can have more than one answer set. Suppose that we have one seat and two persons  $p$  and  $q$ , and we want to assign the seat to a person. We can model this by the following program  $P$

$$\begin{aligned} \text{seat}(p) &\leftarrow \text{not seat}(q). \\ \text{seat}(q) &\leftarrow \text{not seat}(p). \end{aligned}$$

It has two answer sets  $S_1 = \{\text{seat}(p)\}$  and  $S_2 = \{\text{seat}(q)\}$ . Indeed, by applying the Gelfond-Lifschitz transformation we can obtain the reduct program  $P^{S_1}$  consisting of the only rule  $\text{seat}(p) \leftarrow$ . The second rule is removed in the first step of the transformation, and the body of the first rule is removed in the second step.  $P^{S_1}$  is a naf-free program, and has a unique answer set  $\{\text{seat}(p)\}$ . This answer set coincides with our hypothesis  $S_1$ , which means that it is an answer set of the initial program  $P$ . One can verify in a similar way that  $S_2$  is an answer set too.

**Definition 5.5** (Answer set). *An interpretation  $I$  is an answer set of a ground program  $P$  iff it is an answer set of the reduct program  $P^I$ .*

In the remainder, we denote the set of all answer sets of a program  $P$  as  $\text{AS}(P)$ .

### 5.3 Time-dependent programs

The number of ASP applications is growing fast (we refer to Chapter 6 for the in-depth discussion on the examples in the domain of biology). Some of these approaches require an adaptation of the general-purpose solving process to their

specific needs to allow for faster answer set computation. One domain of ASP applications uses programs that depend on a time parameter that bounds the size of a solution. In this section, we define these programs formally and propose a technique to solve the subset of these programs efficiently in Section 5.4.

In the remainder of this chapter, we designate certain predicates as *time-dependent predicates* and denote atoms built with these predicates as

$$p(t_1, \dots, t_{n-1})@ \theta$$

where  $\theta$  is called a *time argument*. This is a convenience notation that separates the (semantic) notion of time from the underlying syntactic representation. This notation is translated to a conventional atom of the form  $p(t_1, \dots, t_{n-1}, \theta)$  at grounding time.

**Definition 5.6** (Time-dependent program). A time-dependent program  $\mathbf{P}$  is a tuple  $\langle P, \tau \rangle$  over a signature  $\sigma = \langle \gamma, v, \pi \rangle$ , such that  $P$  is an answer set program over  $\sigma$  and  $\tau \subseteq \pi$  is a set of time-dependent predicates. We denote the set of  $n$ -ary time-dependent predicates as  $\tau_n$ . We define the set of free time-dependent literals

$$\mathcal{F}_{\mathbf{P}} = \bigcup \left\{ \{p(t_1, \dots, t_{n-1})@ \theta, \neg p(t_1, \dots, t_{n-1})@ \theta\} \mid \begin{array}{l} t_1, \dots, t_{n-1} \in \gamma \cup v \cup \epsilon, \\ \theta \in v \cup \epsilon, p \in \tau_n \end{array} \right\}$$

and the set of bound time-dependent literals

$$\mathcal{B}_{\mathbf{P}} = \bigcup \left\{ \{p(t_1, \dots, t_{n-1})@ \theta, \neg p(t_1, \dots, t_{n-1})@ \theta\} \mid \begin{array}{l} t_1, \dots, t_{n-1} \in \gamma \cup v \cup \epsilon, \\ \theta \in \gamma, p \in \tau_n \end{array} \right\}.$$

The literals from  $\mathcal{F}_{\mathbf{P}}$  contain a variable or a variable expression as the time argument, while the literals from  $\mathcal{B}_{\mathbf{P}}$  contain a constant as the time argument. The set of time-dependent literals of a time-dependent program  $\mathbf{P}$  is defined as  $Lit(\mathbf{P})^\tau = \mathcal{F}_{\mathbf{P}} \cup \mathcal{B}_{\mathbf{P}}$ . Furthermore, for  $l \in Lit(\mathbf{P})^\tau$ , we use  $t_{arg}(l)$  to refer to the time argument  $\theta$  of  $l$ . A time-dependent program  $\mathbf{P}$  is called well-typed iff

$$\forall r \in P \cdot (Lit(\mathbf{P})^\tau \cap (pos(r) \cup neg(r)) \neq \emptyset) \Rightarrow (head(r) \in Lit(\mathbf{P})^\tau)$$

Intuitively, if a rule in a well-typed time-dependent program contains a time-dependent literal in its body, it should contain a time-dependent literal in its head. In the remainder we will only consider well-typed time-dependent programs.

Consider e.g. the following time-dependent answer set program, for which the grounding size depends on the parameter  $t_{max}$ .

$t_{max}$	Answer set
0	$A = \{r(str), time(0), p(0)\}$
1	$B = \{r(str), time(0), p(0), time(1), p(1), q(1)\}$
2	$C = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2)\}$ $D = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2)\}$
3	$E = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), v(3)\}$ $G = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), w(3)\}$ $F = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), w(3)\}$ $H = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), v(3)\}$
4	$I = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), v(3), time(4), p(4), q(4), v(4)\}$ $J = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), w(3), time(4), p(4), q(4), v(4)\}$ $K = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), w(3), time(4), p(4), q(4), v(4)\}$ $L = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), v(3), time(4), p(4), q(4), v(4)\}$ $M = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), v(3), time(4), p(4), q(4), w(4)\}$ $N = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), w(3), time(4), p(4), q(4), w(4)\}$ $O = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), w(3), time(4), p(4), q(4), w(4)\}$ $Q = \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), v(3), time(4), p(4), q(4), w(4)\}$

Table 5.1: Answer sets of the program from Example 5 for different time bounds

**Example 5** Time-dependent program  $\mathbf{P} = \langle P, \{v, w, q, p, time\} \rangle$  consists of the following rules:

$$\begin{aligned}
 & time(0 \dots t_{max}). \\
 q@T & \leftarrow p@(T-1), time(T), time(T-1). \\
 v@T & \leftarrow q@(T-1), not\ w@T, time(T), time(T-1). \\
 w@T & \leftarrow q@(T-1), not\ v@T, r(X), time(T), time(T-1). \\
 p@T & \leftarrow time(T). \\
 & r(str).
 \end{aligned}$$

where  $time(0 \dots t_{max})$  is a shorthand for the facts  $time@0, \dots, time@t_{max}$ . This well-typed program describes the behaviour of a system whose properties depend on time. The answer sets for this program change as the time boundary  $t_{max}$  increases, as illustrated in Table 5.1. When  $t_{max} = 0$  there is only one answer set  $A$ , for  $t_{max} = 1$  the unique answer set is  $B$ , which is twice the size of the previous one. For  $t_{max} = 2$ , negation as failure comes into play, resulting in two different answer sets  $C$  and  $D$ . For  $t_{max} = 3$  there are already four different answer sets, and this number will double with every next time step. As this example illustrates, the number of answer sets as well as the size of the answer sets of a time-dependent program can increase combinatorially in the time boundary.

An important task when modelling and simulating a time-dependent system is to find its steady states. Answer set  $E$  in Table 5.1 contains a steady state of the system described by program  $P$ , as  $time, p, q$ , and  $v$  (and no other time-dependent predicates) belong to  $E$  both for time step 2 and 3.

The main problem in finding these steady states is that it is typically not known in advance at what time steps they occur. Furthermore a system may converge to several steady states, e.g. in answer sets

$$E = \{r(str), time(0), p(0), time(1), p(1), q(1), \underline{time(2)}, \underline{p(2)}, \underline{q(2)}, \underline{v(2)}, \underline{time(3)}, \underline{p(3)}, \underline{q(3)}, \underline{v(3)}\}$$

and

$$G = \{r(str), time(0), p(0), time(1), p(1), q(1), \underline{time(2)}, \underline{p(2)}, \underline{q(2)}, \underline{w(2)}, \underline{time(3)}, \underline{p(3)}, \underline{q(3)}, \underline{w(3)}\}$$

in Table 5.1, we can observe different steady states formed by the underlined predicates. Alternatively, it may oscillate among several states repeatedly, e.g. a

cycle that manifests itself in answer set  $Q$  for  $t_{max} = 4$

$$Q = \{r(\underline{str}), \underline{time}(0), p(0), \underline{time}(1), p(1), q(1), \underline{time}(2), \underline{p}(2), \underline{q}(2), \underline{w}(2), \\ \underline{time}(3), p(3), q(3), v(3), \underline{time}(4), \underline{p}(4), \underline{q}(4), \underline{w}(4))\}$$

in Example 5 is composed of states, formed by underlined predicates. When analyzing a system behaviour, we may want to find all such steady states and cycles. A brute force approach of estimating a time upper bound and grounding and solving the program w.r.t. that upper bound may lead to a suboptimal solving time: if the upper bound is estimated too high, the grounded program is larger than necessary to find the steady states, hence requiring unnecessary work, and if it is estimated too low, not all steady states are found, meaning the process needs to be redone for a larger estimate.

Next, we define concepts related to time-dependent programs that will be useful for devising an efficient solving procedure.

**Definition 5.7** (*t*-grounding of a time-dependent literal). *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a time-dependent program and  $t \in \mathbb{N}$ . The *t*-grounding of a literal  $l \in Lit(\mathbf{P})$ , denoted as  $Gnd(l)_t$ , is obtained as follows: 1) if  $l \in Lit(\mathbf{P}) \setminus \mathcal{F}_{\mathbf{P}}$  then  $Gnd(l)_t = l$ ; 2) if  $l \in \mathcal{F}_{\mathbf{P}}$  then the variable in  $t_{arg}(l)$  is replaced by  $t$ , and in case of a variable expression the resulting value is calculated. In all cases, the obtained literal  $Gnd(l)_t$  is subsequently translated to the conventional ASP notation. For a set of literals  $L$ , we define the *t*-grounding of this set as  $Gnd(L)_t = \bigcup_{l \in L} Gnd(l)_t$ , i.e. we take the pointwise *t*-grounding of its elements.*

**Example 6** The 2-grounding of literal  $l = p(X, a)@(T + 1)$  is  $Gnd(l)_2 = p(X, a, 3)$ .

**Definition 5.8** (*t*-grounding of a rule). *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a time-dependent program and  $t \in \mathbb{N}$ . The *t*-grounding of a rule  $r \in P$  is defined as*

$$Gnd(r)_t = Gnd(head(r))_t \leftarrow Gnd(pos(r))_t, not\ Gnd(neg(r))_t.$$

**Definition 5.9** ( $t_{max}$ -grounding of a time-dependent program). *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a time-dependent program and  $t_{max} \in \mathbb{N}$ . The  $t_{max}$ -grounding of  $\mathbf{P}$  is defined as*

$$Gnd(\mathbf{P})_{t_{max}} = Gnd(\{Gnd(r)_{t'} \mid r \in P, t' \in \mathbb{N}, t' \leq t_{max}\})$$



Intuitively, to obtain  $Gnd(\mathbf{P})_{t_{max}}$  we instantiate all time-dependent literals with a set of time points  $\{t' | 0 \leq t' \leq t_{max}\}$  and then ground the resulting program in the conventional way.

**Example 7** Let  $\mathbf{P} = \langle P, \{v, w, q, p, time\} \rangle$  where  $P$  is the program from Example 5. Its 2-grounding  $Gnd(\mathbf{P})_2$  is defined as

```

1 : time(0).
2 : time(1).
3 : time(2).
4 : q(0)      ← p(-1), time(0), time(-1).
5 : q(1)      ← p(0), time(1), time(0).
6 : q(2)      ← p(1), time(2), time(1).
7 : v(0)      ← q(-1), not w(0), time(0), time(-1).
8 : v(1)      ← q(0), not w(1), time(1), time(0).
9 : v(2)      ← q(1), not w(2), time(2), time(1).
10 : w(0)     ← q(-1), not v(0), r(str), time(0), time(-1).
11 : w(1)     ← q(0), not v(1), r(str), time(1), time(0).
12 : w(2)     ← q(1), not v(2), r(str), time(2), time(1).
13 : p(0)     ← time(0).
14 : p(1)     ← time(1).
15 : p(2)     ← time(2).
16 : r(str).
```

Here we omit the grounded versions of rules that include literals  $r(0), r(1), r(2)$  for conciseness. These rules do not contribute to answer sets anyway, since these literals are not defined in the program.

**Definition 5.10** (State of an answer set). Let  $\mathbf{P} = \langle P, \tau \rangle$  be a time-dependent program and  $A$  be an answer set of the  $t_{max}$ -grounding  $Gnd(\mathbf{P})_{t_{max}}$  for  $t_{max} \in \mathbb{N}$ . Furthermore let  $t \in \mathbb{N}$  with  $t \leq t_{max}$ . The state of  $A$  at time point  $t$  is defined as

$$A^t = \{l \mid l \in A, t_{arg}(l) = t\}$$

Intuitively, the state of answer set  $A$  of  $Gnd(\mathbf{P})_{t_{max}}$  at time point  $t$  is the set of ground time-dependent literals in  $A$  that were grounded with  $t$  in the time argument. Two states are called equivalent if the only difference between literals in these states is in the values of the time points (see Example 10). We denote state equivalence as  $A^{t'} =_{time} A^{t''}$ .

**Example 8** Consider the program  $Gnd(\mathbf{P})_2$  from Example 7. The answer sets of this program are  $C$  and  $D$  as defined in Table 5.1. The states of answer set  $C$  at time points 0, 1 and 2 are  $C^0 = \{time(0), p(0)\}$ ,  $C^1 = \{time(1), p(1), q(1)\}$  and  $C^2 = \{time(2), p(2), q(2), v(2)\}$ .

**Definition 5.11** (Trajectory of an answer set). *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a time-dependent program and  $A$  an answer set of the  $t_{max}$ -grounding  $Gnd(\mathbf{P})_{t_{max}}$  for  $t_{max} \in \mathbb{N}$ . The trajectory of  $A$  is defined as*

$$T^A = \langle A^0 \dots A^{t_{max}} \rangle$$

**Example 9** The trajectory of answer set  $C$  of program  $Gnd(\mathbf{P})_2$  from Example 7 is

$$T^C = \langle \{time(0), p(0)\}, \{time(1), p(1), q(1)\}, \{time(2), p(2), q(2), v(2)\} \rangle$$

**Definition 5.12** (Steady state, steady cycle). *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a time-dependent program and  $A$  be an answer set of the  $t_{max}$ -grounding  $Gnd(\mathbf{P})_{t_{max}}$  for  $t_{max} \in \mathbb{N}$ . The state of  $A$  at time point  $t$ , with  $t < t_{max}$ , is called a steady state iff  $A^t =_{time} A^{t+1}$ . The sequence  $\langle A^k \rangle_{t_1 \leq k \leq t_2}$ , with  $t_1 \in \mathbb{N}$ ,  $t_2 \in \mathbb{N}$  and  $t_1 < t_2 \leq t_{max}$ , is called a steady cycle iff  $A^{t_1} =_{time} A^{t_2}$ .*

Note that to define whether a state is a steady state it is enough to check the next state, because if it does not change in the next step it will not change in the following steps as well due to the deterministic nature of the network model.

**Example 10** The 3-grounding  $Gnd(\mathbf{P})_3$  of  $\mathbf{P} = \langle P, \{v, w, q, p, time\} \rangle$  where  $P$  is the program from Example 5, has answer sets  $E, F, G$ , and  $H$  as defined in Table 5.1. The states of answer set  $E$  are  $E^0 = \{time(0), p(0)\}$ ,  $E^1 = \{time(1), p(1), q(1)\}$ ,  $E^2 = \{time(2), p(2), q(2), v(2)\}$ , and finally  $E^3 = \{time(3), p(3), q(3), v(3)\}$ .  $E^2$  is a steady state, as  $E^2 =_{time} E^3$ .

**Example 11** The 4-grounding  $Gnd(\mathbf{P})_4$  of  $\mathbf{P} = \langle P, \{v, w, q, p, time\} \rangle$  where  $P$  is the program from Example 5, has answer sets  $I, J, K, L, M, N, O$  and  $Q$  as

defined in Table 5.1. The states of answer set  $Q$  are  $Q^0 = \{time(0), p(0)\}$ ,  $Q^1 = \{time(1), p(1), q(1)\}$ ,  $Q^2 = \{time(2), p(2), q(2), q(2)\}$ ,  $Q^3 = \{time(3), p(3), q(3), v(3)\}$  and  $Q^4 = \{time(4), p(4), q(4), w(4)\}$ . Sequence  $\langle Q^2, Q^3, Q^4 \rangle$  is a steady cycle, as  $Q^2 =_{time} Q^4$ .

When solving time-dependent programs, one is usually interested in finding steady states, steady cycles and trajectories leading to these states, as they can help to verify the model's correctness and/or provide new hypotheses about the behaviour of the underlying system. An important problem is that it is in general impossible to accurately estimate an upper time bound  $t_{max}$  that suffices to find all steady states. Thus, one should manually adjust the bound and recompute answer sets over and over, which is very inefficient. In the following section we narrow down time-dependent programs to Markovian programs and propose an approach that does not require a time bound estimation for trajectory computation.

## 5.4 Markovian programs

In this section we define a subclass of time-dependent programs, called Markovian programs. This type of time-dependent programs is defined in such a way that every next state directly depends only on the previous state, and does not depend on any of the future states (hence the name Markovian). This is a reasonable assumption as real-world models are normally unaware of any future events and make their decisions based on the information directly available. We illustrate the use of Markovian programs for biological regulatory network modelling further in Chapter 6.

Recall that steady states and steady cycles for a time-dependent program  $P$  can be found by grounding the program for a manually chosen time upper bound  $t_{max}$  (see Definition 5.9), solving the resulting ground program  $Gnd(P)_{t_{max}}$  to obtain its answer sets, and verifying whether the answer sets reveal steady states or cycles (see Definition 5.12). The Achilles' heel in this procedure is in the manual choice of  $t_{max}$ . Iteratively incrementing it and repeating the above process until reaching a time point  $t_{max}$  at which a steady state or cycle is encountered is inefficient, because that would require solving  $Gnd(P)_0$ ,  $Gnd(P)_1$ ,  $Gnd(P)_2$ , ...,  $Gnd(P)_{t_{max}}$ , or, in other words, grounded versions of the original time-dependent program for time intervals  $\{0, 1\}$ ,  $\{0, 1, 2\}$ , ...,  $\{0, \dots, t_{max}\}$ . Instead, we propose to consecutively solve smaller programs for intervals  $\{0, 1\}$ ,  $\{1, 2\}$ , ...,  $\{t_{max} - 1, t_{max}\}$ . This approach is more efficient be-

cause we ground only for one time step at a time and solve smaller programs in every iteration. Further in this section we show that by doing so we obtain the same answer sets as by solving the initial program for interval  $\{0, \dots, t_{max}\}$ .

**Definition 5.13** (Markovian program). *A time-dependent program  $\mathbf{P} = \langle P, \tau \rangle$  is called Markovian iff it satisfies the following conditions for every  $r \in P$  with  $head(r) \in Lit(\mathbf{P})^\tau$  and  $t \in \mathbb{N}$ :*

1.  $t_{arg}(head(r)) \in \gamma \cup v$
2. for all  $l \in Lit(r) \cap Lit(\mathbf{P})^\tau$  it holds that
  - (a)  $t_{arg}(Gnd(head(r))_t) = t_{arg}(Gnd(l)_t)$  or
  - (b)  $t_{arg}(Gnd(head(r))_t) = t_{arg}(Gnd(l)_t) + 1$

The first constraint in the definition, which says that the head literal may contain only a constant or a variable as a time argument, is purely syntactical and intended to simplify the formal treatment below. Note that we can do this without loss of generality, as a rule in a time-dependent program with a head of the form  $l@T + k$  can be transformed by subtracting  $k$  from every time argument in the rule. The second constraint essentially says that if the head of a rule depends on time point  $t$  then the time-dependent literals in its body may depend only on time point  $t$  or  $t - 1$ , i.e., the current state of the system only depends on its previous state and does not depend on any earlier states. As can be seen in the definition, Markovian programs are a proper subset of time-dependent programs, and all definitions for time-dependent programs hold for Markovian programs as well. Rules in a Markovian program  $\mathbf{P}$  can be divided into two subsets: a program that describes temporal relationships  $P^\tau = \{r \mid r \in P, Lit(r) \cap Lit(\mathbf{P})^\tau \neq \emptyset\}$  and a program that describes the rest of the relationships  $P^e = P \setminus P^\tau$ . Program  $P^e$  can be interpreted as environmental conditions that are invariant over time. By definition,  $P^e$  is independent from the program's temporal part, thus it can be solved separately to obtain its answer sets that represent the values of these conditions.

**Example 12** Consider Markovian program  $\mathbf{P}$  as defined in Example 5. Here the program  $P^\tau$  contains the first five rules, while the program  $P^e$  contains the last rule.

**Definition 5.14** (Partial temporal grounding). *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a Markovian program and  $t \in \mathbb{N}$ . The partial temporal grounding of  $\mathbf{P}$  for time point  $t$  is*

defined as

$$P_t = \{Gnd(r)_t \mid r \in P, head(r) \in Lit(\mathbf{P})^\tau, t_{arg}(Gnd(head(r)))_t = t\}$$

In other words, the partial temporal grounding for a time point  $t$  is the set of  $t$ -grounded rules whose head depends on time point  $t$ .

**Example 13** The partial temporal grounding of  $\mathbf{P}$  built in Example 7 for time point 2 is the program  $P_2$  that is defined as follows

$$\begin{array}{ll} 3 : & time(2). \\ 6 : & q(2) \quad \leftarrow \quad p(1), time(2), time(1). \\ 9 : & v(2) \quad \leftarrow \quad q(1), not\ w(2), time(2), time(1). \\ 12 : & w(2) \quad \leftarrow \quad q(1), not\ v(2), r(X), time(2), time(1). \\ 15 : & p(2) \quad \leftarrow \quad time(2). \end{array}$$

Assume that the  $t_{max}$ -grounding  $Gnd(\mathbf{P})_{t_{max}}$  of a Markovian program  $\mathbf{P}$  has an answer set  $A$ . Once  $A$  is known, using Definition 5.10, we can straightforwardly find its states at time points  $0, 1, \dots, t_{max}$ , i.e.,  $A^0, A^1, \dots, A^{t_{max}}$ . Below we show that it is also possible to find states of an answer set without prior knowledge of the answer set itself. In particular, the state  $A^t$  of an (unknown) answer set of  $Gnd(\mathbf{P})_{t_{max}}$  at time point  $t$  can be computed based on knowledge of the state  $A^{t-1}$  at time point  $t-1$ , as well as knowledge of an answer set  $A^{-1}$  of  $Gnd(P^e)$ . This means that from the answer sets of  $Gnd(P^e)$ , the set of states at time point 0 can be found, and from this the set of states at time point 1, etc. This is done by building  $P'_t = Gnd(P_t \cup \{l \leftarrow . \mid l \in A^{t-1} \cup A^{-1}\})$  and transforming it by replacing literals from  $A^{t-1} \cup A^{-1}$  with true values, which is formally defined in Definition 5.15. Solving the resulting (small) program yields as answer sets the possible states at time point  $t$  given the state  $A^{t-1}$  and the environmental conditions  $A^{-1}$ .

**Definition 5.15** (Partial reduct). *Let  $P$  be a ground program,  $I$  an interpretation of  $P$  and  $P_I = \{l \leftarrow . \mid l \in I\}$  such that  $P_I \subseteq P$  and  $head(P \setminus P_I) \cap I = \emptyset$ . The partial reduct of  $P$  w.r.t.  $I$  is the program  $R^I(P)$  defined as*

$$R^I(P) = \{head(r) \leftarrow (pos(r) \setminus I), not\ neg(r). \mid r \in P \setminus P_I, neg(r) \cap I = \emptyset\}$$

Intuitively,  $I$  is the information we know a priori, but do not want to include in the answer set of a program  $P$ . Partial reduct grounds  $P$  with respect to this knowledge, and remove this prior information before computing answer sets.

**Example 14** Assume we known that  $\{time(1), p(1), q(1)\}$  is the state at time point 1 of a (possibly unknown) answer set of program  $Gnd(\mathbf{P})_2$  from Example 7. We also know an answer set of  $P^e$ , namely  $\{r(str)\}$ . Let  $P_2$  be the partial temporal grounding of  $\mathbf{P}$  for time point 2 as described in Example 13. We construct the set  $I = \{time(1), p(1), q(1)\} \cup \{r(str)\}$  and the program  $P'_2 = Gnd(P_2 \cup \{l \leftarrow . | l \in I\})$  as follows:

```

time(1).
time(2).
p(1).
q(1).
r(str).
q(2)      ←  p(1), time(2), time(1).
v(2)      ←  q(1), not w(2), time(2), time(1).
w(2)      ←  q(1), not v(2), r(str), time(2), time(1).
p(2)      ←  time(2).

```

Adding the literals from  $I$  as facts to program  $P_2$  is necessary in order to ground the rules in  $P_2$  consistently with the full grounding of  $\mathbf{P}$ . For example, without adding the literal  $r(str) \in I$  to  $P_2$  it would not be possible to obtain the grounding of rule 12 in this program, because variable  $X$  would not be grounded with constant  $str$  in this rule.

The partial reduct  $R^I(P'_2)$  is then defined as

```

time(2).
q(2)      ←  time(2).
v(2)      ←  not w(2), time(2).
w(2)      ←  not v(2), time(2).
p(2)      ←  time(2).

```

By applying the partial reduct we remove the literals from  $I$  that appear positively in rule bodies as well as the facts that appear as literals in  $I$ . The answer sets of the resulting program are  $\{time(2), p(2), q(2), v(2)\}$  and  $\{time(2), p(2), q(2), w(2)\}$  which correspond to  $C^2$  and  $D^2$  with  $C$  and  $D$  as in Table 5.1.

The theorem below states that instead of computing answer sets of the  $t_{max}$ -grounding of a Markovian program  $Gnd(\mathbf{P})_{t_{max}}$  directly, we can compute answer sets of smaller programs for every time step  $0 \leq t' \leq t_{max}$  consecutively and obtain the same result. This fact has as the important implication that we

can arrive at answer sets of a Markovian program without considering  $t_{max}$  at all, which makes possible to impose not only a stable state observation, but arbitrary network state as the termination condition. The technique sketched in the example above forms the basis for an algorithm for computing steady states explained in the following section.

**Theorem 5.4.1.** *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a Markovian program and  $Gnd(\mathbf{P})_{t_{max}}$  be the  $t_{max}$ -grounding of  $\mathbf{P}$  for  $t_{max} \in \mathbb{N}$ , then*

$$AS(Gnd(\mathbf{P})_{t_{max}}) = \left\{ \bigcup_{t=-1}^{t_{max}} B^t \mid \begin{array}{l} B^{-1} \in AS(P^e), \\ B^t \in AS(R^{B^{t-1} \cup B^{-1}}(P'_t)) \end{array} \right\}$$

with  $P'_t = Gnd(P_t \cup \{l \leftarrow . \mid l \in B^{t-1} \cup B^{-1}\})$ .

Note that in the definition of  $P'_t$  we add facts from  $B^{t-1}$  and from  $B^{-1}$  to the partial temporal grounding  $P_t$ . These facts are further removed by taking the partial reduct of  $P'_t$  w.r.t  $B^{t-1} \cup B^t$ , but this addition is needed to properly ground  $P_t$ . To prove Theorem 5.4.1 we first formulate and prove the following lemma.

**Lemma 5.4.2.** *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a Markovian program and  $Gnd(\mathbf{P})_{t_{max}}$  be the  $t_{max}$ -grounding of  $\mathbf{P}$  for  $t_{max} \in \mathbb{N}$ . Let  $A \in AS(Gnd(\mathbf{P})_{t_{max}})$ . As usual, let  $A^t \subseteq A$  denote the state of  $A$  at time point  $t$ , for  $t \in \mathbb{N}$  with  $t \leq t_{max}$ , and let  $A^{-1} \subseteq A$  denote the set of literals from  $A$  that are not time dependent. Let  $P_t$  be the partial temporal grounding of  $\mathbf{P}$  for time point  $t$  and*

$$P'_t = Gnd(P_t \cup \{l \leftarrow . \mid l \in A^{t-1} \cup A^{-1}\})$$

*be a ground program obtained from  $P_t$ . Then for all  $t \in \mathbb{N}$  with  $t \leq t_{max}$*

$$A^t \in AS(R^{A^{t-1} \cup A^{-1}}(P'_t))$$

*Proof.* According to Definition 5.4 we need to show that  $A^t$  is a minimal model of

$$X = \{head(r) \leftarrow pos(r) \mid r \in R^{A^{t-1} \cup A^{-1}}(P'_t), A^t \cap neg(r) = \emptyset\}$$

Let us first write down some facts we know about the rules of  $R^{A^{t-1} \cup A^{-1}}(P'_t)$ . From Definition 5.14 (partial temporal grounding) and Definition 5.15 (partial reduct) it follows that for every rule  $r \in R^{A^{t-1} \cup A^{-1}}(P'_t)$ :

$$t_{arg}(head(r)) = t \quad (5.2)$$

$$pos(r) \cap (A^{t-1} \cup A^{-1}) = \emptyset \quad (5.3)$$

$$neg(r) \cap (A^{t-1} \cup A^{-1}) = \emptyset \quad (5.4)$$

Furthermore, for every rule  $r \in R^{A^{t-1} \cup A^{-1}}(P'_t)$  there is a corresponding rule  $r' \in Gnd(\mathbf{P})_{t_{max}}$  with

$$head(r) = head(r') \quad (5.5)$$

$$pos(r) = pos(r') \setminus (A^{t-1} \cup A^{-1}) \quad (5.6)$$

$$neg(r) = neg(r') \quad (5.7)$$

From (5.2) and (5.5) it follows that  $t_{arg}(head(r')) = t$ . Then, from Definition 5.13 (Markovian program) it follows that

$$\begin{aligned} &\text{The literals in the body of } r' \text{ depend on time } t \text{ or } t-1 \\ &\text{or are not time dependent at all.} \end{aligned} \quad (5.8)$$

Note that, using (5.6) in the second equivalence below,

$$\begin{aligned} pos(r) \subseteq A^t &\equiv (\forall l)(l \in pos(r) \Rightarrow l \in A^t) \\ &\equiv (\forall l)(l \in pos(r') \setminus (A^{t-1} \cup A^{-1}) \Rightarrow l \in A^t) \\ &\equiv (\forall l)(l \in pos(r') \wedge l \notin A^{t-1} \cup A^{-1} \Rightarrow l \in A^t) \\ &\equiv (\forall l)(l \in pos(r') \Rightarrow l \in A^{t-1} \cup A^{-1} \vee l \in A^t) \\ &\equiv (\forall l)(l \in pos(r') \Rightarrow l \in A^t \cup A^{t-1} \cup A^{-1}) \\ &\equiv pos(r') \subseteq A^t \cup A^{t-1} \cup A^{-1} \end{aligned}$$

from which we can conclude that

$$pos(r) \not\subseteq A^t \equiv pos(r') \not\subseteq A^t \cup A^{t-1} \cup A^{-1} \quad (5.9)$$

First we show that  $A^t$  is a model of  $X$ .

To show that  $A^t$  is a model of  $X$  we need to show that all rules of  $X$  are satisfied w.r.t.  $A^t$ , i.e., for every  $r \in R^{A^{t-1} \cup A^{-1}}(P'_t)$ :

$$A^t \cap neg(r) = \emptyset \Rightarrow pos(r) \not\subseteq A^t \vee head(r) \in A^t \quad (5.10)$$



Below we prove the implication (5.10) for  $r \in R^{A^{t-1} \cup A^{-1}}(P'_t)$ :

$$\begin{aligned}
& A^t \cap \text{neg}(r) = \emptyset \\
\Rightarrow & \langle (5.4) \rangle \\
& (A^t \cup A^{t-1} \cup A^{-1}) \cap \text{neg}(r) = \emptyset \\
\Rightarrow & \langle r' \text{ is the rule from } Gnd(\mathbf{P})_{t_{max}} \text{ that corresponds to } r; (5.7) \rangle \\
& (A^t \cup A^{t-1} \cup A^{-1}) \cap \text{neg}(r') = \emptyset \\
\Rightarrow & \langle \text{Definition 5.10 (state of an answer set); (5.8)} \rangle \\
& A \cap \text{neg}(r') = \emptyset \\
\Rightarrow & \langle A \text{ is an answer set of } Gnd(\mathbf{P})_{t_{max}} \rangle \\
& \text{pos}(r') \not\subseteq A \vee \text{head}(r') \in A \\
\Rightarrow & \langle \text{Definition 5.10 (state of an answer set); (5.5); (5.2)} \rangle \\
& \text{pos}(r') \not\subseteq A^t \cup A^{t-1} \cup A^{-1} \vee \text{head}(r) \in A^t \\
\Rightarrow & \langle (5.9) \rangle \\
& \text{pos}(r) \not\subseteq A^t \vee \text{head}(r) \in A^t
\end{aligned}$$

Thus,  $A^t$  is a model of  $X$ .

Next, we show that  $A^t$  is a minimal model of  $X_t$ , where  $X_t$  is a partial temporal grounding of  $X$  for a time point  $t$ . We prove it by contradiction. Assume there exists a model  $B$  of  $X_t$  with  $B \subset A^t$ . Consider

$$A' = A^{-1} \cup A^0 \cup \dots \cup A^{t-1} \cup B \cup A^{t+1} \cup \dots \cup A^{t_{max}} \quad (5.11)$$

Also keep in mind that, because of Definition 5.10 (state of an answer set),

$$A = A^{-1} \cup A^0 \cup \dots \cup A^{t-1} \cup A^t \cup A^{t+1} \cup \dots \cup A^{t_{max}} \quad (5.12)$$

Thus,  $A' \subset A$ . According to Definition 5.4 (Gelfond-Lifschitz reduct):

$$Gnd(\mathbf{P})_{t_{max}}^A = \{\text{head}(r') \leftarrow \text{pos}(r') \mid r' \in Gnd(\mathbf{P})_{t_{max}}, A \cap \text{neg}(r') = \emptyset\}$$

where  $Gnd(\mathbf{P})_{t_{max}}^A$  is a reduct of  $Gnd(\mathbf{P})_{t_{max}}$  w.r.t.  $A$ . If we show that  $A'$  is a model of reduct  $Gnd(\mathbf{P})_{t_{max}}^A$  then  $A$  is not a minimal model of this reduct and thus not an answer set of  $Gnd(\mathbf{P})_{t_{max}}$ , which contradicts the formulation of the lemma. This would mean that our assumption about the existence of  $B$  is invalid, and hence  $A^t$  is a minimal model of  $X_t$ .

To show that  $A'$  is a model of  $Gnd(\mathbf{P})_{t_{max}}^A$  we need to show that for every  $r' \in Gnd(\mathbf{P})_{t_{max}}$

$$A \cap \text{neg}(r') = \emptyset \Rightarrow \text{pos}(r') \not\subseteq A' \vee \text{head}(r') \in A'$$

For rules  $r' \in Gnd(\mathbf{P})_{t_{max}}$  that do not contain time-dependent literals we have

$$\begin{aligned}
 & A \cap neg(r') = \emptyset \\
 \Rightarrow & \langle A \text{ is a model of } Gnd(\mathbf{P})_{t_{max}} \rangle \\
 & pos(r') \not\subseteq A \vee head(r') \in A \\
 \Rightarrow & \langle 5.12; r \text{ does not contain time-dependent literals} \rangle \\
 & pos(r') \not\subseteq A^{-1} \vee head(r') \in A^{-1} \\
 \Rightarrow & \langle 5.11; r \text{ does not contain time-dependent literals} \rangle \\
 & pos(r') \not\subseteq A' \vee head(r') \in A'
 \end{aligned}$$

For time-dependent rules  $r' \in Gnd(\mathbf{P})_{t_{max}}$  we distinguish between the following two cases:

$$1. \ t_{arg}(head(r')) = k \text{ with } k \in \{0, 1, \dots, t-1, t+1, \dots, t_{max}\}.$$

$$2. \ t_{arg}(head(r')) = t$$

Note that  $A \cap neg(r') = \emptyset$  implies  $(A^{k-1} \cup A^{-1}) \cap neg(r') = \emptyset$  for all  $k \in \{0, 1, \dots, t_{max}\}$ , hence the existence of a rule  $r \in R^{A^{k-1} \cup A^{-1}}(P'_k)$  corresponding to  $r'$ .

Proof for the first case:

$$\begin{aligned}
 & A \cap neg(r') = \emptyset \\
 \Rightarrow & \langle (5.12) \rangle \\
 & A^k \cap neg(r') = \emptyset \\
 \Rightarrow & \langle (5.7); r \in R^{A^{k-1} \cup A^{-1}}(P'_k) \text{ is the rule that corresponds to } r' \rangle \\
 & A^k \cap neg(r) = \emptyset \\
 \Rightarrow & \langle \text{definition of } X_k \rangle \\
 & head(r) \leftarrow pos(r) \in X_k \\
 \Rightarrow & \langle A^k \text{ is a model of } X_k \rangle \\
 & pos(r) \not\subseteq A^k \vee head(r) \in A^k \\
 \Rightarrow & \langle (5.5); (5.9) \rangle \\
 & pos(r') \not\subseteq (A^k \cup A^{k-1} \cup A^{-1}) \vee head(r') \in A^k \\
 \Rightarrow & \langle (5.8); A^k \subseteq A' \text{ since } k \neq t \rangle \\
 & pos(r') \not\subseteq A' \vee head(r') \in A'
 \end{aligned}$$

Proof for the second case:

$$\begin{aligned}
& A \cap \text{neg}(r') = \emptyset \\
\Rightarrow & \langle (5.12) \rangle \\
& A^t \cap \text{neg}(r') = \emptyset \\
\Rightarrow & \langle (5.7); r \in R^{A^{t-1} \cup A^{-1}}(P'_t) \text{ is the rule that corresponds to } r' \rangle \\
& A^t \cap \text{neg}(r) = \emptyset \\
\Rightarrow & \langle \text{definition of } X_t \rangle \\
& \text{head}(r) \leftarrow \text{pos}(r). \in X_t \\
\Rightarrow & \langle B \text{ is a model of } X_t \text{ by assumption} \rangle \\
& \text{pos}(r) \not\subseteq B \vee \text{head}(r) \in B \\
\Rightarrow & \langle (5.5); (5.9) \rangle \\
& \text{pos}(r') \not\subseteq (B \cup A^{t-1} \cup A^{-1}) \vee \text{head}(r') \in B \\
\Rightarrow & \langle (5.8); B \subseteq A' \rangle \\
& \text{pos}(r') \not\subseteq A' \vee \text{head}(r') \in A'
\end{aligned}$$

□

To proceed, we recall the notions related to splitting sets and sequences, introduced by Lifschitz and Turner [95].

**Definition 5.16** (Splitting set). *A splitting set for a program  $P$  is any set  $U$  of literals such that, for every rule  $r \in P$ , if  $\text{head}(r) \cap U \neq \emptyset$  then  $\text{Lit}(r) \subseteq U$ . The set of rules  $r \in P$  such that  $\text{Lit}(r) \subseteq U$  is denoted as  $\text{bot}_U(P)$ , and the set of rules  $P \setminus \text{bot}_U(P)$  is denoted as  $\text{top}_U(P)$ .*

**Definition 5.17** (Partial evaluation). *The partial evaluation of a program  $P$  with splitting set  $U$  with respect to a set of literals  $X$  is the program  $\text{eval}_U(P, X) = \{\text{head}(r) \leftarrow \text{pos}(r) \setminus U, \text{not}(\text{neg}(r) \setminus U). \mid r \in P, \text{pos}(r) \cap U \subseteq X, \text{neg}(r) \cap U \cap X = \emptyset\}$ .*

**Definition 5.18** (Solution). *Let  $U$  be a splitting set of  $P$ . A solution to  $P$  with respect to  $U$  is a pair  $\langle X, Y \rangle$  of literals such that*

- $X$  is an answer set of  $\text{bot}_U(P)$
- $Y$  is an answer set of  $\text{eval}_U(\text{top}_U(P), X)$

**Theorem 5.4.3** (Splitting theorem). *Let  $U$  be a splitting set for a program  $P$ . A set  $S$  of literals is an answer set of  $P$  iff  $S = X \cup Y$  for some solution  $\langle X, Y \rangle$  to  $P$  with respect to  $U$ .*

**Example 15** Consider a program  $P$

$$\begin{array}{lcl} 1 : & a & \leftarrow \text{not } b. \\ 2 : & b & \leftarrow c, \text{not } a \\ 3 : & c. & \end{array}$$

The set  $U = \{c\}$  splits  $P$  such that  $\text{bot}_U(P)$  contains rule 3 and  $\text{top}_U(P)$  contains rules 1 and 2. The answer set of  $\text{bot}_U(P)$  is  $A = \{c\}$ . The partial evaluation of  $\text{top}_U(P)$  with respect to  $A$  is then defined as  $\text{eval}_U(P, A) = \{a \leftarrow \text{not } b. b \leftarrow \text{not } a\}$ . The answer sets of  $\text{eval}_U(P, A)$  are sets  $\{a\}$  and  $\{b\}$ , and the possible solutions of  $P$  with respect to  $U$  are  $\langle \{c\}, \{a\} \rangle$  and  $\langle \{c\}, \{b\} \rangle$ , that indeed correspond to the answer sets of  $P$   $\{c, a\}$  and  $\{c, b\}$ .

Further on, we will need another lemma:

**Lemma 5.4.4.** *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a Markovian program and  $\text{Gnd}(\mathbf{P})_{t_{\max}}$  be the  $t_{\max}$ -grounding of  $\mathbf{P}$  for  $t_{\max} \in \mathbb{N}$ . Let  $\text{Gnd}(P^e) = \text{Gnd}(P \setminus P^\tau)$ , i.e., the program that contains all the non time-dependent rules from  $P$ . Let  $A \in \text{AS}(\text{Gnd}(\mathbf{P})_{t_{\max}})$  and let  $A^{-1} \subseteq A$  denote the set of literals from  $A$  that are not time-dependent, then  $A^{-1} \in \text{AS}(\text{Gnd}(P^e))$ .*

*Proof.* Let us consider  $U_{-1} = \text{Lit}(\text{Gnd}(P^e))$ .  $U_{-1}$  is a splitting set for program  $\text{Gnd}(\mathbf{P})_{t_{\max}}$  by Definition 5.13 (Markovian program) and Definition 5.16 (splitting set). Furthermore,  $\text{bot}_{U_{-1}}(\text{Gnd}(\mathbf{P})_{t_{\max}}) = \text{Gnd}(P^e)$ .

According to Theorem 5.4.3 (Splitting theorem) answer set  $A = X_{-1} \cup X_0$  where according to Definition 5.18 (Solution)  $X_{-1} \in \text{AS}(\text{bot}_{U_{-1}}(\text{Gnd}(\mathbf{P})_{t_{\max}}))$  and  $X_0 \in \text{AS}(\text{eval}_{U_{-1}}(\text{top}_{U_{-1}}(\text{Gnd}(\mathbf{P})_{t_{\max}}), X_{-1}))$ . Note that  $X_0$  does not contain non time-dependent literals, because due to Definition 5.13 (Markovian program) every rule in program  $\text{eval}_{U_{-1}}(\text{top}_{U_{-1}}(\text{Gnd}(\mathbf{P})_{t_{\max}}), X_{-1})$  has a time-dependent literal as its head. On the other hand,  $X_{-1}$  does not contain time-dependent literals since program  $\text{bot}_{U_{-1}}(\text{Gnd}(\mathbf{P})_{t_{\max}})$  does not contain time-dependent literals. Thus,  $X_{-1} = A^{-1}$ . Since  $X_{-1}$  is an answer set of  $\text{bot}_{U_{-1}}(\text{Gnd}(\mathbf{P})_{t_{\max}})$  and  $\text{bot}_{U_{-1}}(\text{Gnd}(\mathbf{P})_{t_{\max}}) = \text{Gnd}(P^e)$ , this concludes the proof that  $A^{-1} \in \text{AS}(\text{Gnd}(P^e))$   $\square$

**Corollary 5.1.** *Let  $\mathbf{P} = \langle P, \tau \rangle$  be a Markovian program and  $\text{Gnd}(P^e) = \text{Gnd}(P \setminus P^\tau)$ . If  $\text{Gnd}(P^e)$  does not have an answer set, then for any  $t_{\max} \in \mathbb{N}$ ,  $\text{Gnd}(\mathbf{P})_{t_{\max}}$  does not have an answer set either.*

Next, we introduce the notion of ordinals and sequences that is necessary to introduce the concept of splitting sequences described by Lifschitz and Turner [95].

The finite *ordinals* are the nonnegative integers. The first ordinal 0 is defined by the empty set,  $\emptyset$ . The second ordinal  $1 = \{0\} = \{\emptyset\}$ . The third ordinal  $2 = \{0, 1\} = \{\emptyset, \{\emptyset\}\}$  and so forth. The first *infinite ordinal* is the set of natural numbers  $\omega = \{0, 1, 2, \dots\}$ . The successor of an ordinal  $\alpha$  is  $\alpha + 1 = \alpha \cup \{\alpha\}$ . The ordinal other than 0 that is not a successor of any other ordinal is called a *limit ordinal*.  $\omega$  is the first limit ordinal.

A sequence  $\langle U_\alpha \rangle_{\alpha < \mu}$  is a family whose index set is an initial segment of ordinals  $\{\alpha : \alpha < \mu\}$  where  $\mu$  is the length of the sequence. A sequence is *monotone* if  $U_\alpha \subseteq U_\beta$  for  $\alpha < \beta$  and *continuous* if, for each limit ordinal  $\alpha < \mu$ ,  $U_\alpha = \bigcup_{\eta < \alpha} U_\eta$ .

**Definition 5.19** (Splitting sequence). A splitting sequence for a program  $P$  is a monotone continuous sequence  $\langle U_\alpha \rangle_{\alpha < \mu}$  of splitting sets for  $P$  such that  $\bigcup_{\alpha < \mu} U_\alpha = \text{Lit}(P)$ .

**Definition 5.20** ([95]). Let  $U = \langle U_\alpha \rangle_{\alpha < \mu}$  be a splitting sequence for a program  $P$ . A solution to  $P$  with respect to  $U$  is a sequence  $\langle X_\alpha \rangle_{\alpha < \mu}$  of sets of literals such that

- $X_0$  is an answer set of  $\text{bot}_{U_0}(P)$ ,
- for any ordinal  $\alpha$  such that  $\alpha + 1 < \mu$ ,  $X_{\alpha+1}$  is an answer set of the program  $\text{eval}_{U_\alpha}(\text{bot}_{U_{\alpha+1}}(P) \setminus \text{bot}_{U_\alpha}(P), \bigcup_{\nu \leq \alpha} X_\nu)$ ,
- for any limit ordinal  $\alpha < \mu$ ,  $X_\alpha = \emptyset$ , and
- $\bigcup_{\alpha \leq \mu} X_\alpha$  is consistent

**Theorem 5.4.5** (Splitting sequence theorem [95]). Let  $U = \langle U_\alpha \rangle_{\alpha < \mu}$  be a splitting sequence for a program  $P$ . A set  $S$  of literals is an answer set of  $P$  iff  $S = \bigcup_{\alpha < \mu} X_\alpha$  for some solution  $\langle X_\alpha \rangle_{\alpha < \mu}$  to  $P$  with respect to  $U$ .

**Example 16** Consider program  $P$  for Example 15. A sequence  $\langle \{c\}, \{c, a\} \rangle$  is a splitting sequence since  $\{c\}$  is an answer set of  $\text{bot}_{\{c\}}(P)$  and  $\{c, a\}$  is an answer set of  $\text{eval}_{\{c, a\}}(\text{bot}_{\{c, a\}}(P) \setminus \text{bot}_{\{c\}}(P), \{c, a\})$ .

Now we can prove Theorem 5.4.1. Let us repeat the formulation here again:

**Theorem 5.4.1.** Let  $\mathbf{P} = \langle P, \tau \rangle$  be a Markovian program and  $\text{Gnd}(\mathbf{P})_{t_{max}}$  be

the  $t_{max}$ -grounding of  $\mathbf{P}$  for  $t_{max} \in \mathbb{N}$ , then

$$AS(Gnd(\mathbf{P})_{t_{max}}) = \left\{ \bigcup_{t=-1}^{t_{max}} B^t \mid \begin{array}{l} B^{-1} \in AS(Gnd(P^e)), \\ B^t \in AS(R^{B^{t-1} \cup B^{-1}}(P'_t)) \end{array} \right\}$$

with  $P'_t = Gnd(P_t \cup \{l \leftarrow . \mid l \in B^{t-1} \cup B^{-1}\})$ .

*Proof.* Let us denote the right hand side of the formula in the theorem formulation as  $C$ . To prove the theorem in the right direction we show that  $AS(Gnd(\mathbf{P})_{t_{max}}) \subseteq C$ . Consider  $A \in AS(Gnd(\mathbf{P})_{t_{max}})$ . Obviously,  $A = A^{-1} \cup \bigcup_{t=0}^{t_{max}} A^t$ , where  $A^{-1}$  is an answer set of  $Gnd(P^e)$  according to Lemma 5.4.4. It is now sufficient to show that for  $0 \leq t \leq t_{max}$  we have  $A^t \in AS(R^{A^{-1} \cup A^{t-1}}(P'_t))$ , which follows immediately from Lemma 5.4.2.

To prove the theorem in the left direction, we show that  $C \subseteq AS(Gnd(\mathbf{P})_{t_{max}})$  using splitting sequences.

Let  $U_{-1} = Lit(Gnd(P^e))$ , then by Definition 5.13 (Markovian program) and Definition 5.16 (splitting set)  $U_{-1}$  is a splitting set for program  $Gnd(\mathbf{P})_{t_{max}}$ .

Let  $U_t = Lit(Gnd(\mathbf{P})_t)$  with  $0 \leq t \leq t_{max}$ , i.e., the set of literals from  $\mathbf{P}$  that is grounded up to time point  $t$ . Then by Definition 5.13 (Markovian program) and Definition 5.16 (splitting set)  $U_t$  is a splitting set for  $Gnd(Gnd(\mathbf{P})_{t_{max}})$ . Note that  $U_{t-1} \subseteq U_t$  for  $0 \leq t \leq t_{max}$ .

According to Definition 5.13 (Markovian program)  $bot_{U_t}(Gnd(\mathbf{P})_{t_{max}})$  includes all rules  $r$  such that  $head(r)$  is non time-dependent or  $t_{arg}(head(r)) \leq t$ , and  $top_{U_t}(Gnd(\mathbf{P})_{t_{max}})$  includes all rules  $r$  such that  $t_{arg}(head(r)) > t$  for  $0 \leq t \leq t_{max}$ .

Since  $U_{t'} \subseteq U_t$  for  $t' < t$  and  $\bigcup_{t=-1}^{t_{max}} U_t = Lit(Gnd(\mathbf{P})_{t_{max}})$  then by Definition 5.19 (splitting sequence)  $U = \langle U_t \rangle_{-1 \leq t \leq t_{max}}$  is a splitting sequence for program  $Gnd(\mathbf{P})_{t_{max}}$ .

We now show that for every  $B \in C$  with  $B = B^{-1} \cup \dots \cup B^t \cup \dots \cup B^{t_{max}}$  it holds that  $\langle B^t \rangle_{-1 \leq t \leq t_{max}}$  is a solution to  $Gnd(\mathbf{P})_{t_{max}}$  w.r.t.  $U$ . Using Theorem 5.4.5 it then follows that  $B$  is indeed an answer set of  $Gnd(\mathbf{P})_{t_{max}}$ .

In line with Definition 5.20, first we need to show that  $B^{-1}$  is an answer set of  $bot_{U_{-1}}(Gnd(\mathbf{P})_{t_{max}})$ . This holds because  $bot_{U_{-1}}(Gnd(\mathbf{P})_{t_{max}}) = Gnd(P^e)$  and, by definition,  $B^{-1} \in AS(Gnd(P^e))$ .

Next, we need to show that for  $t \leq t_{max}$

$$B^t \in AS(eval_{U_{t-1}}(bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t_{max}}), \bigcup_{t' < t} B^{t'}))$$

Note that the program  $bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t_{max}})$  contains all rules which have  $t$  as a time argument in the head, and no other rules. Therefore, by Definition 5.13 (Markovian program) we can write that

$$eval_{U_{t-1}}(bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t_{max}}), \bigcup_{t' < t} B^{t'}) = \\ eval_{U_{t-1}}(bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t_{max}}), B^{-1} \cup B^{t-1})$$

Now, by Definition 5.15 (partial reduct) and Definition 5.17

$$eval_{U_{t-1}}(bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t_{max}}), B^{-1} \cup B^{t-1}) = \\ R^{B^{t-1} \cup B^{-1}}(P'_t)$$

By definition,  $B^t \in AS(R^{B^{t-1} \cup B^{-1}}(P'_t))$  and hence

$$B^t \in AS(eval_{U_{t-1}}(bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t_{max}}), \bigcup_{t' < t} B^{t'}))$$

Next, we need to show that for any  $t > t_{max}$

$$AS(eval_{U_{t-1}}(bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t-1}), \bigcup_{t' < t} B^{t'})) = \emptyset$$

Note, that for any  $t > t_{max}$  program

$$eval_{U_{t-1}}(bot_{U_t}(Gnd(\mathbf{P})_{t_{max}}) \setminus bot_{U_{t-1}}(Gnd(\mathbf{P})_{t-1}), \bigcup_{t' < t} B^{t'})$$

is empty, as  $Gnd(\mathbf{P})_{t_{max}}$  is grounded only till time point  $t_{max}$  and thus does not contain any literals that depend on  $t$ . Thus, for any  $t > t_{max}$  this program has a single answer set  $\{\emptyset\}$ .

Finally,  $\bigcup_{t \leq t_{max}} B^t$  is consistent because every  $B^t$  is consistent for  $t \leq t_{max}$  (since it is an answer set), and  $B^t \cap B^{t'} = \emptyset$  for  $t \neq t'$  by Definition 5.10.

We have shown that  $\langle B^t \rangle_{-1 \leq t \leq t_{max}}$  is a solution to  $Gnd(\mathbf{P})_{t_{max}}$  w.r.t.  $U$  which concludes the proof.  $\square$

## 5.5 Discovery of steady states

Assume that we have a Markovian program that models the behaviour of a system. The practical question of interest is to find the steady states and cycles

observed in the answer sets of an ASP program that models a biological system, and all trajectories that lead to these steady states and cycles. To solve this problem we propose the *Temporal Algorithm*. This algorithm explores the space of solutions in a breadth-first search manner. It starts with all possible combinations of environmental conditions (answer sets of  $P^e$ ), continuously increments the time boundary and computes the answer sets of a program, which correspond to changing states of a biological model at the current time point. The algorithm maintains the list of trajectories  $\mathcal{T}$  that is updated with every new time step and a list of ‘active’ states  $\mathcal{A}$ , which the system has reached on the current time step. Eventually, some trajectories converge to steady states or cycles, and the states that correspond to the current states of these trajectories are removed from  $\mathcal{A}$  so that they are not considered in the following steps. Eventually, all trajectories will reach steady states or cycles and  $\mathcal{A}$  will become empty, which terminates the algorithm. The list of trajectories will contain all trajectories that have been found by the algorithm, every trajectory finishing with a steady state or cycle. The pseudocode for this algorithm is shown below.

**Require:**  $\mathbf{P} = \langle P, \tau \rangle$ ; {A Markovian program}

- 1:  $\mathbf{V} = \emptyset$ ; {The set of visited states}
- 2:  $\mathcal{A}_{-1} = \text{List}(\text{AS}(P^e))$ ; {The list of sets of initial conditions that will be used for grounding}
- 3:  $n = \text{size}(\mathcal{A}_{-1})$ ;
- 4:  $\mathcal{A} = \text{List}(\mathcal{A}_{-1}[k] | 1 \leq k \leq n)$ ; {The list of sets of current states that are rooted in the  $k$ -th set of initial conditions}
- 5:  $\mathcal{T} = \mathcal{A}_{-1}$ ; {The list of trajectories}
- 6:  $t = 0$ ;
- 7: **repeat**
- 8:   Obtain the partial temporal grounding  $P_t$  of  $\mathbf{P}$ ;
- 9:   **for**  $k = 1$  to  $n$  **do**
- 10:      $A^{-1} = \mathcal{A}_{-1}[k]$ ; {Obtain the  $k$ -th set of initial conditions}
- 11:      $\mathbf{A}_{\text{new}} = \emptyset$ ;
- 12:     {Iterate over the sets of current states rooted in the  $k$ -th set of initial conditions}
- 13:     **for all**  $\mathbf{A} \in \mathcal{A}[k]$  **do**
- 14:       {Iterate over the current states rooted in the  $k$ -th set of initial conditions}
- 15:       **for all**  $A^{t-1} \in \mathbf{A}$  **do**
- 16:           $P'_t = \text{Gnd}(P_t \cup \{l \leftarrow . | l \in A^{t-1} \cup A^{-1}\})$ ; {Grounding  $P_t$ }
- 17:           $S = \text{AS}(R^{A^{t-1} \cup A^{-1}}(P'_t))$ ; {Obtaining states for time point  $t$ }



---

```

18:   for all  $A^t \in S$  do
19:     Add state  $A^t$  to all trajectories in  $\mathcal{T}$  that contain  $A^{t-1}$ 
20:     if  $A^t$  is not in  $\mathbf{V}$  then
21:       Add state  $A^t$  to the set  $\mathbf{V}$  of visited states
22:       Add state  $A^t$  to the set  $\mathbf{A}_{new}$  of states that should be consid-
       ered in the following iteration
23:     end if
24:   end for
25: end for
26: end for
27:  $\mathcal{A}[k] = \mathbf{A}_{new}$ ;
28: end for
29:  $t = t + 1$ ;
30: until  $\mathcal{A}$  is empty or  $\mathcal{A}$  contains only empty sets
31: return  $\mathcal{T}$ 

```

**Example 17** Let us consider the program from Example 5 and look how the Temporal Algorithm is applied to this program.

- We initialize  $\mathbf{V} = \emptyset$ ,  $\mathcal{A}_{-1} = [\{r(str)\}]$ ,  $\mathcal{A} = [\{\{r(str)\}\}]$  and  $\mathcal{T} = [\{r(str)\}]$ . In this case  $n = 1$  since  $P^e$  has only one answer set.
- After finding the states for time point 0
  - $\mathcal{A} = [\{\{time(0), p(0)\}\}]$
  - $\mathcal{T} = [\{r(str), time(0), p(0)\}]$
  - $\mathbf{V} = \{\{time(0), p(0)\}\}$
- After the next step for time point 1
  - $\mathcal{A} = [\{\{time(1), p(1), q(1)\}\}]$
  - $\mathcal{T} = [\{r(str), time(0), p(0), time(1), p(1), q(1)\}]$
  - $\mathbf{V} = \{\{time(0), p(0)\}, \{time(1), p(1), q(1)\}\}$
- After the next step for time point 2
  - $\mathcal{A} = [\{\{time(2), p(2), q(2), v(2)\}, \{time(2), p(2), q(2), w(2)\}\}]$
  - $\mathcal{T} = [\{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2)\}, \{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2)\}]$
  - $\mathbf{V} = \{\{time(0), p(0)\}, \{time(1), p(1), q(1)\}, \{time(2), p(2), q(2), v(2)\}, \{time(2), p(2), q(2), w(2)\}\}$
- After the next step for time point 3
  - $\mathcal{A} = [\emptyset]$

- $\mathcal{T} = [\{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), v(3)\},$   
 $\{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), w(3)\},$   
 $\{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), v(2), time(3), p(3), q(3), w(3)\},$   
 $\{r(str), time(0), p(0), time(1), p(1), q(1), time(2), p(2), q(2), w(2), time(3), p(3), q(3), v(3)\}]$
- $\mathbf{V} = \{\{time(0), p(0)\}, \{time(1), p(1), q(1)\}, \{time(2), p(2), q(2), v(2)\}, \{time(2), p(2), q(2), w(2)\}\}$

A graphical representation of the algorithm flow and the obtained trajectories is shown in Figure 5.5.1. Here the expanded execution tree is shown for every iteration of the outermost loop of the algorithm. Highlighted leaves correspond to the states of the answer set of the Markovian program at time point  $t$ , i.e., to the content of  $\mathcal{A}$ . In the first three iterations of the outermost loop new states are discovered, while the fourth iteration (for  $t = 3$ ) leads only to already visited states. Every path from the root of the tree to a leaf corresponds to a trajectory in  $\mathcal{T}$ . In case when  $P^e$  has more than one answer set we would start with a separate tree for every answer set of  $P^e$ , but these trees may further have some branches in common. The final tree may be collapsed to represent a state space of the program as it is shown on the right side of Figure 5.5.1. By keeping the set of trajectories  $\mathcal{T}$  and the set of visited states  $\mathbf{V}$ , this algorithm is capable to find all steady states and cycles of the modelled network. If the system arrives again at one of the states that have already been passed (line 19), the state is not included into  $\mathbf{A}_{new}$  and thus not considered at following steps. Using this algorithm we do not need to specify the number of time steps any more, which solves the problems stated in the beginning of this chapter.

## 5.6 Related work

In this chapter we have proposed a method to find all steady states of a Markovian program efficiently. However, our approach is not the only way to deal with the problem. Gebser et al. have recently proposed an incremental program solving approach and a specially constructed solver *iclingo* that allows for solving incremental programs [60]. Even though this solver, when used for the Markovian programs that we introduced in this chapter, terminates as soon as the first

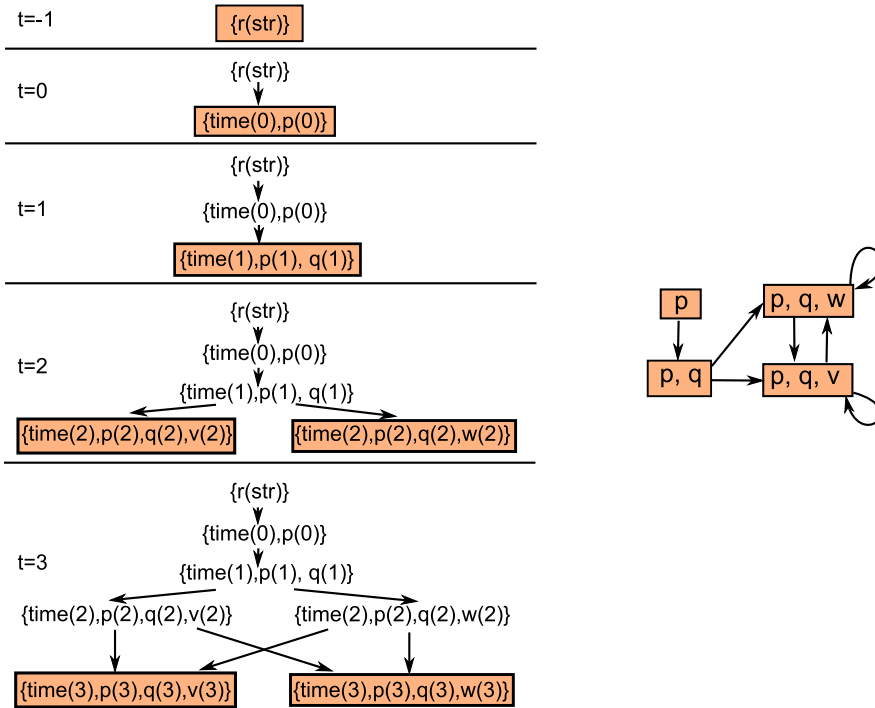


Figure 5.5.1: An illustration of the Temporal Algorithm execution for the program from Example 5

steady state is encountered, and hence unlike our approach does not find all steady states, Gebser et al.'s proposal is relevant to our work. Here we provide a more detailed description of the incremental program solving approach and highlight the differences with our proposal.

An incremental program in the sense of [60] includes a special incremental parameter  $k$  and consists of three parts: base, cumulative and volatile. Each part contains ASP rules that conform to certain restrictions. The base part does not use the incremental parameter anywhere in the rules; it contains constant (with respect to  $k$ ) information. The cumulative part can use parameter  $k$  in the head of rules and parameter  $(k - n), n \in \mathbb{N}$  in the body of rules. This part accumulates the information that is being generated by the incremental program during solving. The volatile part, which can be thought of as a query to the program, can use parameter  $k$ , both in the body and in the head of rules. This part essentially contains constraints imposed on an answer set, and as soon as these constraints are satisfied, the solving process is terminated as an answer set is found. During the solving process the base part is grounded only once, as it does not depend on  $k$ . Parameter  $k$  is incremented automatically, and for every value of  $k$  the cumulative part is grounded once, while keeping the rules obtained for previous values of  $k$ . In the volatile part, only the rules that are grounded for the current value of  $k$  are considered. For every value of  $k$ , the solver tries to solve the program obtained by merging all three parts together, and the process terminates as soon as an answer set is found. The advantage of this approach compared to the usual solving process is that it reduces the effort of computing the answer set for unknown  $k$ . In the common answer set program solving setting, a user would be forced to compute the grounding for every  $k$  until an answer set is found. This results in multiple groundings of the base and cumulative part, which is avoided in the incremental setting.

If we regard the incremental parameter  $k$  as time, we can simulate a Markovian program  $\mathbf{P} = \langle P, \tau \rangle$  by putting  $P^e$  in the base part and  $P^\tau$  in the cumulative part. However, implementing the volatile part is not straightforward. Given the set  $\tau$  of time-dependent predicates we can write rules to capture steady states or cycles and define a constraint over the occurrence of such a state or cycle in the volatile part, as illustrated below.

**Example 18** Let  $\mathbf{P} = \langle P, \tau \rangle$  be a Markovian program over a signature  $\sigma = \langle \gamma, v, \pi \rangle$  and  $\tau = \{u, v\}$  where  $u, v \in \pi$  are unary time-dependent predicates. We define an incremental program  $P'$  from  $\mathbf{P}$  as explained above, i.e. by putting  $P^e$  in the base part of  $P'$  and  $P^\tau$  in the cumulative part of  $P'$ . The exact contents of  $P^e$  and  $P^\tau$  do not matter for the sake of this example. Next, we add the

following set of rules to the cumulative part of  $P'$ :

$$\begin{aligned}
 & \text{int\_k}(0..k-1). \\
 h(k) & \leftarrow \text{not } u(k), \text{not } u(k-T_1), \text{not } v(k), \text{not } v(k-T_1), \text{int\_k}(T_1). \\
 h(k) & \leftarrow u(k), u(k-T_1), \text{not } v(k), \text{not } v(k-T_1), \text{int\_k}(T_1). \\
 h(k) & \leftarrow v(k), v(k-T_1), \text{not } u(k), \text{not } u(k-T_1), \text{int\_k}(T_1). \\
 h(k) & \leftarrow v(k), v(k-T_1), u(k), u(k-T_1), \text{int\_k}(T_1).
 \end{aligned}$$

In this set of rules we define the termination conditions of the program. As we have two time-dependent predicates in the program, there are four possible states: only  $v$  or only  $u$  is present, both are present, and both are absent. To check whether a given state produces the steady state or cycle at time point  $k$  we need to iterate through all time points in the past, i.e., from 0 to  $k-1$  and to check whether this state was observed before. Every state requires a separate rule that checks the specific combination of predicates for this state, as illustrated in the program above.

Finally, we initialize the volatile part of  $P'$  with the rule  $\leftarrow \text{not } h(k)$ . The appearance of  $h(k)$  in an answer set of  $P'$  indicates that a steady state or cycle is found. The constraint in the volatile part only allows answer sets that contain  $h(k)$ .

However, there are two pitfalls associated with the above encoding. First, the number of rules that needs to be added to the cumulative part grows exponentially with the number and the arity of time-dependent predicates; recall that we do not only need all combinations of time-dependent predicates, but also all their possible groundings. Secondly, the solver terminates as soon as the first steady state is encountered, and hence does not generate all steady states of the program. Applying a meta-procedure similar to the Temporal Algorithm from Section 5.5, i.e., running the incremental program for every possible starting state would provide only a partial solution to this problem, because one starting state may converge to several steady states, and all but one of these states will be omitted in the answer set of an incremental program. For these reasons, the approach we proposed in this chapter is a more suitable candidate to tackle the steady state search problem in Markovian programs.

Action languages [63], another set of formalisms applicable to solve time-dependent programs, provide a high-level description language that can be adopted to model time-dependent systems. However, they suffer from the same drawback as incremental programs: it is not possible to define a set of constraints that allow finding all steady states and cycles. A more elaborated discussion on action languages is presented in the next chapter.

## 5.7 Conclusions

In this chapter we have introduced time-dependent answer set programs, which are useful to model systems whose behaviour depends on time. An important task when modelling such systems is to find their steady states and cycles. Unfortunately, it is typically not known in advance at what time steps these steady states manifest themselves. A brute force approach of estimating a time upper bound and grounding and solving the program w.r.t. that upper bound may lead to a bad solving time: if the upper bound's estimate is too high, the grounded program is larger than necessary to find the steady states, hence requiring unnecessary work, and if it is too low, not all steady states (if any) are found and the process needs to be redone for a larger estimate.

We have proposed an efficient algorithm for solving Markovian programs, i.e., time-dependent programs for which the next state of the program depends only on the previous state of the program. This is a reasonable assumption as real-world models are normally unaware of any future events and make their decisions based on the information directly available. Instead of solving Markovian programs for some long time interval  $\{0, \dots, t_{max}\}$  we consecutively solve smaller programs for intervals  $\{0, 1\}$ ,  $\{1, 2\}$ ,  $\dots$ ,  $\{t_{max} - 1, t_{max}\}$ , which can be done more efficiently. We have shown that by doing so we obtain the same answer sets as by solving the initial program for interval  $\{0, \dots, t_{max}\}$ .

# 6 Modelling biological regulatory networks with Answer Set Programming

---

## 6.1 Introduction

Answer Set Programming (ASP) has applied to various domains ranging from computer games [90] and tourism [78] to code optimization [32] and space shuttle diagnosis [7]. In the field of systems biology ASP also finds its place, as will become clear in this chapter.

ASP is a declarative programming language, i.e., it provides a means to describe a system using a high-level language and to analyze its behaviour and properties. Being able to represent a system in an abstract way is a very appealing feature from the biological perspective. There is a multitude of ways biological entities may interact in the real world, and it is impossible to capture every specific aspect of these interactions in a formal model. ASP provides a theoretically founded approach to abstract away from a specific representation and to express information in terms of general concepts. Of course, one should always keep in mind the tradeoff between the level of abstraction and the actual usefulness of the system – a too abstract representation would not capture any interesting facts, while too many low-level details may complicate the system description. As we will show in this chapter, ASP provides a very good foundation to build such balanced frameworks.

In the last several years there has been an increased interest in using ASP for biological needs. However, approaches to use ASP for biological modelling do

not provide a framework that describes a semantics for system development, which means that for each system the whole program has to be built from scratch and every interaction has to be defined with its own rule. In this chapter we study this problem and propose a unified ASP framework describing a semantics that can be used to build biological regulatory networks.

Inspired by the boolean network modelling approach, this chapter proposes an ASP implementation of the Boolean network semantics that is described in Section 6.4. Moreover, we expand this semantics with new operations that are impossible to model in Boolean networks such as meta-interactions, or the influence of one entity on the interaction between two other entities. Previously created boolean network models can not only be easily represented in our framework, but also structurally simplified as the richer expressivity of our framework allows for more concise (and intuitively understandable) representations.

The main reason for building such network models is to analyze their dynamics, i.e., the way systems evolve over time. As we have discussed in Chapter 5, traditional ASP approaches are not efficient to model network dynamics. In Section 6.5 of this chapter we propose an efficient algorithm to model network dynamics that is based on the ideas developed in Chapter 5. Additionally, we extend the capabilities of the proposed framework by providing a way to not only find the steady states, but also to answer arbitrary queries about the network states and trajectories.

We start the discussion with Section 6.2 where we describe existing ASP approaches that leverage biological network models with reasoning capabilities. Further, in Section 6.3 we informally introduce a new framework to represent threshold boolean networks with ASP, while in Section 6.4 we define this framework more formally and bridge the connection with the theoretical work presented in Chapter 5. In Section 6.5 we discuss efficient modelling techniques and provide the query answering extension for the framework, while in Section 6.6 we describe an evaluation on cell cycle models of budding yeast, of fission yeast and of mammals, which shows the usefulness of our approach and highlights the advantages over standard boolean network representation. Finally, we conclude the chapter in Section 6.7.

## 6.2 Related work

In recent years there have been several attempts to employ ASP to build biological models. One application of ASP in the domain of biology was proposed



by Ray [134]. In this approach the author describes a system called XHAIL that takes a background theory  $B$  and a set of examples  $E$  as an input and provides a hypothesis  $H$  such that  $B \cup H \models E$ . In other words, the system makes a hypothesis how a current system  $B$  can be extended in order to accommodate the observations in  $E$ . To this end the author develops a theoretical approach that is based on abductive and inductive logic programming which is then implemented in ASP. [135] describes the application of XHAIL to build and restore a model of the Aromatic Amino acid pathway of *S. cerevisiae* after corruption with respect to given experimental data. The initial model was built manually, and then different corruption scenarios were evaluated, and in most cases XHAIL was able to restore the initial model structure.

A different approach that deals with network correction using ASP, was proposed by Gebser et al. [59]. Here the authors consider large biological networks constructed from experimental data and describe a technique to automatically repair the networks to assimilate given initial and final observations. The core of this approach is a Sign Consistency Model (SCM) [151] which represents networks in a similar way as threshold boolean networks, described in Section 4.3. SCM postulates that a node can be repressed (activated) if there is at least one repression (activation) link in the graph that points to this node. If the initial and final observations do not conform with a given network, then the network is repaired to assimilate the observations. The repair function may add an input edge, change the type of an edge (from inhibition to activation or vice versa) or activate a node in an initial observation. This approach was evaluated on the *E. coli* transcriptional network and has shown very promising results allowing to repair the network after it had been corrupted in different ways.

XHAIL and the SCM repair model can be crucial for reconstructing a network built from literature and/or experimental data; however they do not deal with network dynamics, i.e., they do not consider the question of how the network state changes over time. Genes and proteins that make up the network influence each other and change their state in time. Analysis of these changes is an important task in systems biology. Below, we review two other approaches that are designed for network dynamics analysis.

These approaches, investigated by several authors [9, 44, 163], are based on action languages. Action languages, first introduced by Gelfond and Lifschitz [64], provide an instrument to describe the behaviour of transition systems and reason about such systems using a high-level language that is internally translated into ASP. Although the notion of time is not present in the action language descriptions explicitly, in the underlying encoding a discrete time component is usually assumed.

The most basic action language  $\mathcal{A}$  contains *fluents* (entities such as genes and proteins) and *actions* (such as interactions) and is defined by three components: a domain description language, an observation language and a query language. The domain description language is used to describe how the actions change the state of fluents by means of rules of the form

$$a \text{ causes } f \text{ if } p_1, \dots, p_n \quad (6.1)$$

where  $a$  is an action and  $f, p_1, \dots, p_n$  are fluent literals. The observation language expresses information about the state of fluents by means of rules of the following form

$$f \text{ after } a_1, \dots, a_n \quad (6.2)$$

$$\text{initially } f \quad (6.3)$$

where  $f$  is a fluent and  $a_1, \dots, a_n$  is a sequence of actions. Intuitively, the first rule says that after executing a given sequence of actions  $f$  will hold, while the second rule is a shorthand of the first rule for the case when the sequence of actions is empty, i.e., it corresponds to the initial state of the system. The query language also contains rules of the form (6.2); contrary to the observation language, in this language they express the goals that we want to evaluate and are thus translated to ASP in a different way. A number of extensions for this action language were proposed, some of which were used to describe biological models.

The first attempt to use action languages for modelling biological systems was made by Baral et al. [9]. In this paper the authors describe an extension for the action language  $\mathcal{A}$ , a language  $\mathcal{A}_T^0$ , that is aimed at expressing biological events such as triggering or inhibiting an interaction. The query language is extended to allow prediction, explanation and planning in biological networks. Prediction queries are used to find out whether a certain fluent is true (a gene is active) in a certain situation in the future with respect to the given initial conditions. Explanation queries are used to find out the state of a fluent in the past, given observations about the current moment. Planning queries are aimed at finding a sequence of actions from the initial state that would allow the system to achieve a given query state. The language was implemented in the BioSigNet system [163]. The system was applied to model the NF $\kappa$ B signal transduction pathway and to reason about the different aspects of this model.

A subsequent application of action languages to model a high-level sulphur starvation response pathway of *Arabidopsis thaliana* was presented in the work

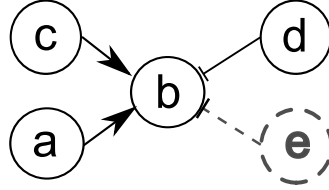


Figure 6.2.1: An illustration for the action language model

of Dworschack et al. [44]. Here the authors take  $\mathcal{A}_T^0$  and another action language  $\mathcal{C}$  and extend it with the concepts of static casual laws and allowance rules in order to be able to express the model of interest. Static casual laws express a relation between fluents that is independent of any particular action, i.e.,

$$f \text{ if } g$$

where  $f$  and  $g$  are the fluents.

An allowance rule says that a particular action  $a$  can happen, however it is not forced to happen in the next time step. Similarly to the approach of Baral et al. [9], prediction, explanation and planning are supported in this language.

These two approaches are similar in nature and provide a means both to describe the model in a high-level language and to analyze its behaviour and properties. However, these approaches do not provide a framework that describes a semantics for model development, which means that for each model the whole program has to be built from scratch and every interaction has to be defined with its own rule. In other words, the framework describes only the description language, and the biologist has to describe every interaction separately. In the approach we propose further in this chapter we go one step further, and provide the biologist with a background theory based on a boolean network model semantics.

Let us consider an example where protein  $b$  is activated by  $a$  and  $c$  and inhibited by  $d$  (solid lines in Figure 6.2.1). In the threshold boolean network semantics, if the number of activating links is higher than the number of inhibition links, then protein  $b$  is activated, if it is lower it is inhibited, if it is equal, the state remains unchanged. To express this with action languages we need two actions: *activating\_prot\_b*, *inhibiting\_prot\_b*, and the following set of statements:

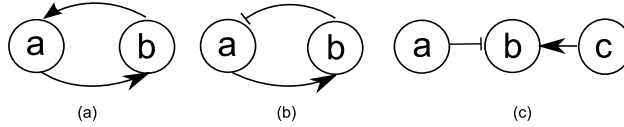


Figure 6.3.1: Examples of Boolean networks

<i>activating_prot_b</i>	<b>causes</b>	<i>b</i>
<i>inhibiting_prot_b</i>	<b>causes</b>	$\neg b$
<i>a, \neg d</i>	<b>causes</b>	<i>activating_prot_b</i>
<i>c, \neg d</i>	<b>causes</b>	<i>activating_prot_b</i>
<i>a, c</i>	<b>causes</b>	<i>activating_prot_b</i>
<i>d, \neg a, \neg c</i>	<b>causes</b>	<i>inhibiting_prot_b</i>

The number of required rules grows combinatorially with the number of interactions, and small changes in the network (e.g. adding an extra node *e* that inhibits *b*) may require many changes in the underlying model. In the following sections we propose a more succinct and modular representation that overcomes these problems.

## 6.3 Boolean networks and ASP

As discussed in Chapter 4 many approaches have been proposed to address the task of biological modelling. Among them, discrete dynamical networks based on Boolean networks are one of the best established qualitative modelling methods that are widely used by biologists to model protein regulatory networks (e.g., [3, 35, 103]). The nodes of a boolean network represent protein molecules and the directed edges represent interactions. Edges can be typed to represent different kinds of interactions, such as inhibition and activation. For example, in Figure 6.3.1a, proteins *a* and *b* activate each other. When at least one of the proteins is active at the initial state, the network settles in the state  $\{act(a), act(b)\}$ , i.e., both proteins will eventually become active.

While liked for their simplicity, dynamical networks have the disadvantage of not being self-descriptive, i.e., they are built under some background assumptions that are not explicitly stated in the network itself. Moreover, dynamical

networks provide little support for reasoning about network behaviour. As argued by Tran [163], reasoning can leverage a biologist's experience and simplify tasks of model analysis and observation assimilation.

In this thesis we propose to represent biological regulatory networks by answer set programs. Example 1 introduces our idea and illustrates the notation we will use throughout the chapter.

**Example 1** Program  $P_1$ , consisting of the rules G1-G6 and S1-S6 below, models the network in Figure 6.3.1a. Rule labels, preceding the rules and separated from them by a colon ':', are introduced to refer to a particular rule as well as to distinguish general rules from specific ones. The specific rules or S-rules describe the structure of a particular network, in this case S1-S4, and initial conditions, in this case S5 and S6. The general rules or G-rules describe the semantics of the network, i.e., what 'activates' or 'inhibits' means in the context of our network. Section 6.4 contains more details.

$G1 :$	$time(0..2).$	
$G2 :$	$act(Y, T)$	$\leftarrow act(X, T - 1), activates(X, Y, T - 1), T > 0.$
$G3 :$	$inh(Y, T)$	$\leftarrow act(X, T - 1), inhibits(X, Y, T - 1), T > 0.$
$G4 :$		$\leftarrow act(X, T), inh(X, T).$
$G5 :$	$act(X, T)$	$\leftarrow act(X, T - 1), not inh(X, T), T > 0.$
$G6 :$	$inh(X, T)$	$\leftarrow inh(X, T - 1), not act(X, T), T > 0.$
$S1 :$	$protein(a).$	
$S2 :$	$protein(b).$	
$S3 :$	$activates(a, b, T).$	
$S4 :$	$activates(b, a, T).$	
$S5 :$	$act(a, 0).$	
$S6 :$	$inh(b, 0).$	

The answer set of  $P_1$  is  $\{act(a, 0), inh(b, 0), act(a, 1), act(b, 1), act(a, 2), act(b, 2)\}$ . Here the predicate  $act(a, 0)$  means that protein  $a$  is active at time 0, and correspondingly,  $inh(b, 0)$  means that protein  $b$  is inhibited at time 0. In this answer set there is no difference between the states at time steps 1 and 2, as both proteins remain active. Hence we conclude that, under the given initial conditions, the steady state of the system is  $\{act(a), act(b)\}$ . As we will see later, the actual answer set of the program includes more information that is not relevant to our task such as the literals built with predicates *protein*, *time*, *activates* and *inhibits*. In our answer set representation we omit this information for the sake of conciseness.

Note that in principle only network specific rules such as rules S1-S6 need to be redefined for a given regulatory network, while the other rules model general biological properties. This makes the representation of such networks as answer set programs intuitively simple, while at the same time the ASP machinery becomes available to analyze and predict the behaviour of the described network at hand.

One of the main advantages of representing regulatory networks by answer set programs is that all background information can be expressed explicitly in the program itself. This allows normalizing differently expressed networks with one standard representation, thus avoiding the ambiguity of different descriptions. Furthermore, the use of ASP eliminates the need for specific network execution algorithms to retrieve the steady states of the networks. In fact, another main advantage of using ASP is that all supporting tools such as solvers and grounders are readily available.

## 6.4 Building a network model in ASP

In this section we set up the framework for describing biological regulatory networks as answer set programs. The framework is presented in two steps. First, we explain the basic set of rules for modelling boolean threshold networks. Second, we extend the basic framework with the features that cannot be modelled by Boolean networks directly. In this way we provide a more elaborate explanation of the rules behind the network as well as reflect the thought process that led to the final implementation.

In Section 6.4.1, we give a detailed explanation of the S-rules and G-rules of  $P_1$  in Example 1. Next we deal with issues such as conflicts, exceptions and self-degradation that do not occur in the network in Figure 6.3.1a, but might manifest themselves in other interaction networks. In Section 6.4.2, we extend the framework with the ability to model complex interactions that target other interactions rather than genes or proteins.

### 6.4.1 Basic framework

#### 6.4.1.1 Describing entities and their influences

The first step in describing a protein network is to introduce the network structure, cfr. rules S1-S6 in Example 1. Rules S1 and S2 define the proteins in the

network, rules S5 and S6 define the initial state of these proteins, while rules S3 and S4 describe activation interactions between proteins. We add the extra time parameter  $T$  in these predicates to be able to model the dynamical network structure. Some interactions can be affected by external factors and be present or absent at different time points.

By themselves these rules (facts) do not model anything; although they define the connection between proteins, they do not describe the influence of these connections on the proteins at different time steps. To this end, we introduce the G-rules.

First of all, rule G1 is merely a shorthand for

$$time(0). \text{ } time(1). \text{ } time(2).$$

to introduce time steps into the program. Rules G2 and G3 define the actual semantics of the activation and inhibition concepts.

Rule G4 is a constraint that says that at any time point  $T$  a protein  $X$  cannot be both active and inhibited at the same time. Rules G5 and G6 are called inertia rules, that are introduced to express the fact that at time point  $T$  a protein  $X$  keeps the same state as at time point  $T - 1$  in case no external influence changes its state at  $T$ .

Once we have described the problem using ASP, the grounder is used to substitute the variables with all possible constants. The activation rule G2, for example, says that protein  $Y$  will be active at time step  $T$  if protein  $X$  is active and there is an activating connection between  $X$  and  $Y$  at the previous time step. When grounded, this rule will result in the following rules:

$$\begin{aligned} act(b, 1) &\leftarrow act(a, 0), activates(a, b, 0), 1 > 0. \\ act(b, 2) &\leftarrow act(a, 1), activates(a, b, 1), 2 > 0. \\ act(b, 1) &\leftarrow act(b, 0), activates(b, b, 0), 1 > 0. \\ act(b, 2) &\leftarrow act(b, 1), activates(b, b, 1), 2 > 0. \\ &\dots \end{aligned}$$

In the programs presented in this chapter, we omit some predicates for the sake of brevity. In the full version of the program every rule that contains variable  $T$  would additionally contain the predicate  $time(T)$ , and every rule that contains at least one of the variables  $X, Y, Z$  would contain a corresponding predicate  $protein(X), protein(Y), protein(Z)$  in its body if not explicitly stated otherwise.

After grounding, the task of an ASP solver is to find an answer set of the ground program. In our application, an answer set contains a sequence of protein states for each time point (e.g., Example 1). Therefore, we can retrieve

the steady state of the network with respect to the transition function implicitly defined by the G-rules by looking at the protein states in an answer set at each time step. When the protein states in two consequent time steps do not change, a steady state has been reached. In Example 1, we reach the steady state at time point 1, because the protein states do not change after this point.

**Example 2** If we change rule S4 in program  $P_1$  to  $inhibits(b, a, T)$ ., we obtain the network from Figure 6.3.1b. The answer set of the resulting program  $P_2$  is  $\{act(a, 0), inh(b, 0), act(a, 1), act(b, 1), inh(a, 2), act(b, 2), inh(a, 3), act(b, 3)\}$  (in the rest of the chapter we omit from answer sets the predicates that are not essential for constructing the trajectory, e.g.,  $protein(a)$ ,  $activates(a, b, 0)$ , etc). Intuitively, this answer set can be explained as follows: all facts are in the answer set by definition, thus the predicates from rules S1-S6 are in the answer set; rules S3 and S5 trigger a ground version of rule G2 that causes the presence of  $act(b, 1)$ ; rule G5 causes the presence of predicate  $act(a, 1)$ ; the fact  $act(b, 1)$  that is already in the answer set together with the new version of rule S4 triggers a ground version of rule G3, resulting in  $inh(a, 2)$ , and rules G5 and G6 result in the other predicates that are in the answer set. From the above we conclude that the steady state of this network is  $\{inh(a), act(b)\}$ .

#### 6.4.1.2 Resolving conflicts

Rules G2-G4 might fail to work for more complex regulatory networks. Below we explain why they should be replaced with more refined rules, as well as supplemented by supporting rules.

**Example 3** Let us consider the network in Figure 6.3.1c. This network can be presented as follows

$S1 : protein(a).$	$S4 : activates(c, b, T).$	$S6 : act(a, 0).$
$S2 : protein(b).$	$S5 : inhibits(a, b, T).$	$S7 : act(b, 0).$
$S3 : protein(c).$		$S8 : act(c, 0).$

The program consisting of these facts S1-S8 together with the rules G1-G6 from Example 1 does not have an answer set under initial conditions S6-S8. Indeed, S5 together with S6 trigger the rule G3, thus forcing  $inh(b, 1)$  to be in the answer set. On the other hand, S4 together with S8 triggers the rule G2, thus pushing  $act(b, 1)$  to the answer set. However, due to constraint G4 these predicates cannot be in the same answer set, thus the program does not have an



answer set at all.

To resolve this conflict, we implement threshold boolean network behaviour as explained in Chapter 4: if there are more incoming activation links than inhibition links, then the protein is active; if there are more inhibition links, then the protein is inhibited; if their number is equal, then the protein keeps the previous state. To implement this, we need to adjust the constraint as well as the activation and inhibition rules. The superscript in the rule labels below denotes the version of the rules.

$$\begin{aligned}
 G2^1 : \text{act}(Y, T) & \leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1), \\
 & \quad A - I > 0, T > 0, \text{int}(A), \text{int}(I). \\
 G3^1 : \text{inh}(Y, T) & \leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1), \\
 & \quad I - A > 0, T > 0, \text{int}(A), \text{int}(I). \\
 G7 : \text{int}(0.. \#\_of\_prot).
 \end{aligned}$$

Rules  $G2^1$  and  $G3^1$  implement the idea behind the threshold network: we count the number of activation and inhibition links for every instance and make the decision based on this count. Literals  $\text{int}(A)$  and  $\text{int}(I)$  define  $A$  and  $I$  to be integers from 0 to the number of proteins defined in the model as stated in rule  $G7$ . The number of incoming edges to any given node cannot be more than the number of the nodes in the network. Rule  $G4$  remains untouched. Note, that when the number of incoming activation and inhibition links is equal, no rules  $G2^1$  or  $G3^1$  are applicable, and the entity remains in the state it was before due to inertia rules  $G5$  and  $G6$ .

The predicates  $\#\_act/3$  and  $\#\_inh/3$  are defined as follows

$$\begin{aligned}
 G8 : \#\_act(Y, A, T) & \leftarrow A \{ \text{act}(X, T) : \text{activates}(X, Y, T) : \text{protein}(X) \} A, \\
 & \quad \text{protein}(Y), \text{int}(A). \\
 G9 : \#\_inh(Y, I, T) & \leftarrow I \{ \text{act}(X, T) : \text{inhibits}(X, Y, T) : \text{protein}(X) \} I, \\
 & \quad \text{protein}(Y), \text{int}(I).
 \end{aligned}$$

The rules  $G8$  and  $G9$  count the number of activation and inhibition links that influence an entity  $Y$  in the current state. The concepts of *conditional literal* and *cardinality constraint* are explained in Section 5.2.

**Example 4** Let us consider the network with 3 entities  $a, b$  and  $c$ , where  $b$  and  $c$  activate  $a$  (i.e. facts  $\text{activates}(b, a, 0)$  and  $\text{activates}(c, a, 0)$  are present). Let us consider the grounding of rule  $G8$  for an entity  $Y = a$  for time point

0, with conditions that  $b$  is active (i.e.  $act(b, 0)$  is present) and  $c$  is inhibited (i.e.  $inh(c, 0)$  is present). The rule G8 then can be grounded as follows

$$\begin{aligned} \#\_act(a, 0, 0) &\leftarrow 0\{act(b, 0), act(c, 0)\}0, protein(a), int(0). \\ \#\_act(a, 1, 0) &\leftarrow 1\{act(b, 0), act(c, 0)\}1, protein(a), int(1). \\ \#\_act(a, 2, 0) &\leftarrow 2\{act(b, 0), act(c, 0)\}2, protein(a), int(2). \\ \#\_act(a, 3, 0) &\leftarrow 3\{act(b, 0), act(c, 0)\}3, protein(a), int(3). \end{aligned}$$

Since, as we said above,  $act(b, 0)$  is present in the program as a fact, while  $act(c, 0)$  is not present, the only rule that is satisfied in this case is

$$\#\_act(a, 1, 0) \leftarrow 1\{act(b, 0), act(c, 0)\}1, protein(a), int(1).$$

because exactly one literal between brackets is satisfied in the program. Thus, the number of activation links for entity  $a$  in this case is 1.

In this way, we solve the problem of resolving conflicting influence by counting the number of activation and inhibition edges that influence a protein in the current network state, as shown further in Example 5.

This setup already allows us to model fairly complex interaction networks, such as the Budding Yeast network [94]. We describe our ASP representation of this network in Section 6.6.

#### 6.4.1.3 Sensitivity thresholds

Some features still cannot be expressed in this framework. For example, proteins can become active when their inhibitors are not active, even without an external activation input. Another example is that some proteins can have a certain ‘tolerance’ to an inhibition/activation influence. For instance, a protein can become inhibited only if two or more proteins that suppress it are active, otherwise it is not affected. To address these issues, we introduce the inhibition and activation thresholds. Let us return to Figure 6.3.1c. Under the current definitions, protein  $b$  does not change its state when both  $a$  and  $c$  are active, i.e., if  $b$  is active it remains active. Imagine now that we want to modify the behaviour of  $b$  to change its sensitivity to the activating or inhibiting influence such that it requires less effort (less activation/inhibition inputs) to change the state of the protein. This requirement can be implemented in the system by introducing inhibition/activation thresholds. The compound numeration in the rule labels

below denotes the supporting rules for the main rule.

$$\begin{aligned}
 G2^2 : \quad act(Y, T) &\leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1), \\
 &\quad act\_th(Y, Th), A - I > Th, T > 0, \\
 &\quad int(A), int(I). \\
 G3^2 : \quad inh(Y, T) &\leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1), \\
 &\quad inh\_th(Y, Th), I - A > Th, T > 0, \\
 &\quad int(A), int(I). \\
 G10 : \quad act\_th(X, 0) &\leftarrow not\ mod\_act\_th(X). \\
 G10.1 : \quad mod\_act\_th(X) &\leftarrow act\_th(X, Th), Th \neq 0. \\
 G11 : \quad inh\_th(X, 0) &\leftarrow not\ mod\_inh\_th(X). \\
 G11.1 : \quad mod\_inh\_th(X) &\leftarrow inh\_th(X, Th), Th \neq 0.
 \end{aligned}$$

Rules that do not change are omitted here. We replace  $G2^1$  and  $G3^1$  by  $G2^2$  and  $G3^2$  so that now they take into account the possible presence of a threshold. Rules G10 and G11 set the activation and inhibition threshold of every protein to 0 in case it was not set explicitly by a special S-rule (G10.1 and G11.1). If the threshold values are not modified, the G-rules described above will lead to the same answer sets as the ones in Section 6.4.1.2. Having both inhibiting and activating thresholds instead of one threshold is not redundant, since these thresholds do not characterize the ‘on/off’ level of the protein, but rather the effort that is needed to change its state. The thresholds can be viewed as tolerance degrees of a protein to a corresponding input. Positive values make the protein more tolerant and negative ones make it less tolerant. The default value can be altered by a specific rule as illustrated in Example 6.

**Example 5** Let  $P_3$  be the answer set program consisting of general rules G1,  $G2^2$ ,  $G3^2$ , G4, G5, G6, G7, G8, G9, G10, G10.1, G11, G11.1 and the specific rules from Example 3 corresponding to Figure 6.3.1c. The activation and inhibition thresholds of these proteins are not explicitly defined; hence they are automatically set to the default value. The answer set of this program is  $\{act(a,0), act(b,0), act(c,0), act(a,1), act(b,1), act(c,1), act(a,2), act(b,2), act(c,2), act(a,3), act(b,3), act(c,3)\}$ . The state of protein  $b$  does not change over time since its inhibiting and activating inputs are equal, and its thresholds for activation and inhibition are both 0. From the answer set we retrieve that the steady state is  $\{act(a), act(b), act(c)\}$ .

**Example 6** For the network in Figure 6.3.1c, let us explicitly set the inhibition

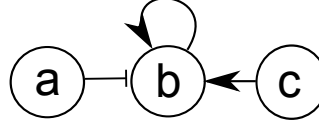


Figure 6.4.1: An example network with a self-activation node

threshold of  $b$  to  $-1$  to indicate that this protein is susceptible to inhibition. In other words, let  $P_4$  be the answer set program containing all the rules from  $P_3$  as well as the additional rule  $S9$  *inh.th*( $b, -1$ ). The answer set of this program is  $\{act(a,0), act(b,0), act(c,0), act(a,1), inh(b,1), act(c,1), act(a,2), inh(b,2), act(c,2), act(a,3), inh(b,3), act(c,3)\}$ . The steady state in this case is  $\{act(a), inh(b), act(c)\}$ .

#### 6.4.1.4 Self-activation and self-degradation

The phenomena of self-activation and self-degradation can also be modelled by adjusting activation and inhibition thresholds. Self-activation/degradation means that a protein is able to change its state when no external influence is applied. To model this type of interaction, we can follow two paths: use thresholds or introduce a loop interaction. These two approaches, although used to model the same phenomenon, exhibit different behaviour.

In the first approach, we set a negative activation (inhibition) threshold on a protein, then this protein will activate (inhibit) itself even without any external influence. Consider the following example

**Example 7** Let  $P_5$  be the answer set program consisting of general rules G1, G2<sup>2</sup>, G3<sup>2</sup>, G4, G5, G6, G7, G8, G9, G10, G10.1, G11, G11.1 and the following specific rules corresponding to Figure 6.3.1c

$S1 : protein(a).$	$S4 : activates(c, b, T).$	$S6 : inh(a, 0).$
$S2 : protein(b).$	$S5 : inhibits(a, b, T).$	$S7 : inh(b, 0).$
$S3 : protein(c).$		$S8 : inh(c, 0).$
$S9 : act.th(b, -1).$		

These rules describe the structure of the network, and reflect the fact that all proteins are inhibited at the initial time point. Moreover, rule S9 sets the activation threshold of  $b$  to  $-1$ , which indicate that  $b$  is susceptible to activation.

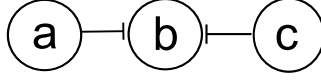


Figure 6.4.2: Node  $b$  is inhibited only if both  $a$  and  $c$  are active

According to rule G2<sup>2</sup>, in this case protein  $b$  activates itself when no inhibition influence is applied, i.e., self-activation takes place. The answer set of program  $P_5$  is  $\{inh(a,0), inh(b,0), inh(c,0), inh(a,1), act(b,1), inh(c,1), inh(a,2), act(b,2), inh(c,2), inh(a,3), act(b,3), inh(c,3)\}$ . The steady state in this case is  $\{inh(a), act(b), inh(c)\}$ .

The second option to model self-activation (self-inhibition) is to change the network structure and introduce an activation (inhibition) edge that points to a protein itself, which is probably a more obvious way of modelling self-activation (self-inhibition). Let us consider the following example

**Example 8** Let  $P'_5$  be the answer set program containing the same rules as program  $P_5$  from Example 7, except rule S9 which is changed to *activates*( $b, b, T$ ). In this case the answer set of program  $P'_5$  is  $\{inh(a,0), inh(b,0), inh(c,0), inh(a,1), inh(b,1), inh(c,1), inh(a,2), inh(b,2), inh(c,2), inh(a,3), inh(b,3), inh(c,3)\}$ . As one can see, in this case behaviour of the network is different from  $P_5$ , because rule S9 does not affect  $b$  while  $b$  is not active. Rather, it will prevent  $b$  from being inhibited when both  $a$  and  $b$  are active, like in Example 6.

#### 6.4.1.5 Encoding exceptions

It may be the case that certain interactions in a biological network model do not conform with the threshold network assumptions and thus cannot be represented within the framework semantics we have provided above. Let us illustrate this with the following example.

**Example 9** Consider the network in Figure 6.4.2 where we want  $b$  to be inhibited only if both  $a$  and  $c$  are active, but not if only one of them is active. This behaviour can be easily captured with the following propositional boolean formula

$$b = \overline{a \wedge c}$$

However, it does not conform to the boolean threshold network semantics and thus cannot be modelled yet within the framework we have proposed so far.

In order to be able to model arbitrary behaviour in the framework we introduce the notion of an ‘abnormal situation’, or exception. This notion can be introduced by replacing  $G2^2$  and  $G3^2$  by

$$\begin{aligned} G2^3 : \quad act(Y, T) &\leftarrow \#\_act(Y, A, T-1), \#\_inh(Y, I, T-1), act\_th(Y, Th), \\ &\quad not\ ab(Y, T-1), A - I > Th, T > 0, int(A), int(I). \\ G3^3 : \quad inh(Y, T) &\leftarrow \#\_act(Y, A, T-1), \#\_inh(Y, I, T-1), inh\_th(Y, Th), \\ &\quad not\ ab(Y, T-1), I - A > Th, T > 0, int(A), int(I). \end{aligned}$$

We modified these rules to include the exceptional behaviour in the framework by means of a predicate  $ab/2$ . Now the state of a gene or protein  $Y$  tagged with predicate  $ab/2$  will not be governed by the framework semantics, and can be redefined according to the user needs.

**Example 10** Consider the situation in Example 9. We build our framework with rules  $G1$ ,  $G2^3$ ,  $G3^3$ ,  $G4$ ,  $G5$ ,  $G6$ ,  $G7$ ,  $G8$ ,  $G9$ ,  $G10$ ,  $G10.1$ ,  $G11$ ,  $G11.1$  and the following S-rules

$$\begin{aligned} S1 : \quad &protein(a). \\ S2 : \quad &protein(b). \\ S3 : \quad &protein(c). \\ S4 : \quad &ab(b, T) \leftarrow both\_active(a, c, T). \\ S5 : \quad &inh(b, T) \leftarrow both\_active(a, c, T-1), T > 0. \\ S6 : \quad &both\_active(a, c, T) \leftarrow act(a, T), act(c, T), T > 0. \end{aligned}$$

Rule S4 says that  $b$  should be processed in an exceptional way in case both  $a$  and  $c$  are active (as defined in rule S6). Rule S5 defines how  $b$  should be processed, i.e., that it should be inhibited in this case. Note that since the facts  $inhibits(a, b, T)$  and  $inhibits(c, b, T)$  are not present in the program, nothing happens with  $b$  when only one of agents  $a$  or  $c$  is active.

The possibility to include a custom-tailored behaviour for every gene or protein makes it easy to model e.g. conventional Boolean networks that have

boolean functions as transition functions on the nodes. As shown by Niemela [116] any propositional boolean formula can be translated to an ASP program, which then can be plugged in as a set of S-rules in the main program. However, excessive use of exceptions may complicate the program code and make it difficult to maintain. Further in Section 6.4.2 we discuss an extension of our framework that will reduce the necessity of such manual adjustments.

#### 6.4.1.6 Starting conditions

By writing S-rules, a user can model various networks and observe their behaviour under certain initial conditions. This requires the user to set various initial protein activation combinations and analyze the results of each execution. On large networks with tens of proteins, the number of different possible combinations is very high, which makes this task very cumbersome. To automate this process, we introduce two additional general rules that deal with different initial condition combinations:

$$\begin{aligned} G12: \quad act(X, 0) &\leftarrow not\ inh(X, 0). \\ G13: \quad inh(X, 0) &\leftarrow not\ act(X, 0). \end{aligned}$$

These rules force the solver to make a choice for each protein: either it is active at the initial time point, or inhibited. In this way, different answer sets are automatically generated for each possible combination of active and inhibited proteins, which drastically decreases the need for manual input.

### 6.4.1.7 Framework summary

To properly conclude Section 6.4.1 we bundle all rules that constitute our modelling framework as follows

$G1 :$	$time(0..2).$	
$G2^3 :$	$act(Y, T)$	$\leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1),$ $act\_th(Y, Th), not\ ab(Y, T - 1), A - I > Th,$ $T > 0, int(A), int(I).$
$G3^3 :$	$inh(Y, T)$	$\leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1),$ $inh\_th(Y, Th), not\ ab(Y, T - 1), I - A > Th,$ $T > 0, int(A), int(I).$
$G4 :$		$\leftarrow act(X, T), inh(X, T).$
$G5 :$	$act(X, T)$	$\leftarrow act(X, T - 1), not\ inh(X, T), T > 0.$
$G6 :$	$inh(X, T)$	$\leftarrow inh(X, T - 1), not\ act(X, T), T > 0.$
$G7 :$	$int(0..\#\_of\_prot).$	
$G8 :$	$\#\_act(Y, A, T)$	$\leftarrow A\{act(X, T) : activates(X, Y, T) :$ $protein(X)\}A, protein(Y), int(A).$
$G9 :$	$\#\_inh(Y, I, T)$	$\leftarrow I\{act(X, T) : inhibits(X, Y, T) :$ $protein(X)\}I, protein(Y), int(I).$
$G10 :$	$act\_th(X, 0)$	$\leftarrow not\ mod\_act\_th(X).$
$G10.1 :$	$mod\_act\_th(X)$	$\leftarrow act\_th(X, Th), Th \neq 0.$
$G11 :$	$inh\_th(X, 0)$	$\leftarrow not\ mod\_inh\_th(X).$
$G11.1 :$	$mod\_inh\_th(X)$	$\leftarrow inh\_th(X, Th), Th \neq 0.$
$G12 :$	$act(X, 0)$	$\leftarrow not\ inh(X, 0).$
$G13 :$	$inh(X, 0)$	$\leftarrow not\ act(X, 0).$

This set of rules is the complete description of the modelling framework built so far. To briefly summarize the distinguishing features of this framework in a single point of reference, in Table 6.1 we compile the descriptions of the examples modelled with this framework that occurred in Section 6.4.1.





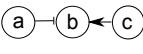
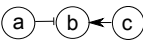
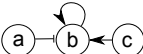
Network	Structure	Steady states	Notes
	<i>protein(a). activates(a, b, T).</i> <i>protein(b). activates(b, a, T).</i>	<i>act(a) act(b).</i> <i>inh(a) inh(b).</i>	Positive feedback
	<i>protein(a). activates(a, b, T).</i> <i>protein(b). inhibits(b, a, T).</i>	<i>inh(a) act(b).</i> <i>inh(a) inh(b).</i>	Negative feedback
	<i>protein(a). inhibits(a, b, T).</i> <i>protein(b). activates(c, b, T).</i> <i>protein(c).</i>	<i>act(a) act(b) act(c).</i> <i>inh(a) act(b) act(c).</i> <i>act(a) inh(b) act(c).</i> <i>inh(a) act(b) inh(c).</i> <i>act(a) inh(b) inh(c).</i> <i>inh(a) inh(b) inh(c).</i>	Conflict
	<i>protein(a). inhibits(a, b, T).</i> <i>protein(b). inhibits(c, b, T).</i> <i>protein(c). act_th(b, -1).</i>	<i>act(a) act(b) act(c).</i> <i>act(a) inh(b) inh(c).</i> <i>inh(a) act(b) inh(c).</i> <i>inh(a) act(b) act(c).</i>	Activation threshold
	<i>protein(a). inhibits(a, b, T).</i> <i>protein(b). inhibits(c, b, T).</i> <i>protein(c). activates(b, b, T).</i>	<i>act(a) act(b) act(c).</i> <i>act(a) act(b) inh(c).</i> <i>act(a) inh(b) act(c).</i> <i>inh(a) act(b) act(c).</i> <i>act(a) inh(b) inh(c).</i> <i>inh(a) act(b) inh(c).</i> <i>inh(a) inh(b) inh(c).</i>	Self-activation

Table 6.1: The summary of examples presented in Section 6.4.1

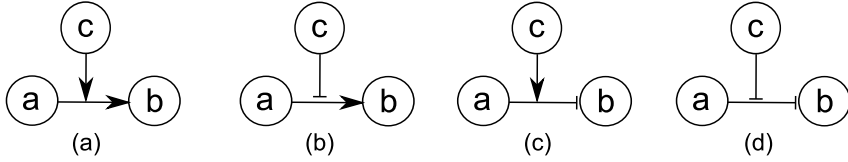


Figure 6.4.3: Additional interaction types for the extended framework

## 6.4.2 Extended framework

The framework described above provides an instrument to model the behaviour of threshold boolean networks. However, the modelling capabilities of this framework are still limited to modelling interactions between genes and proteins only. Consider the networks depicted in Figure 6.4.3, where a protein may influence an interaction. These types of interactions cannot be modelled directly within the boolean networks semantics. It is, however, important to be able to express this type of interactions as they occur fairly often in biological models (see Section 6.6 for examples).

In light of this, it would be desirable to extend our formalism to model the influence of a protein on the interaction between other proteins. As we are concerned with two types of interactions, viz. activation and inhibition, the extension of our framework should handle interactions of the four types depicted in Figure 6.4.3.

### 6.4.2.1 Semantics of interactions

Let us discuss the semantics of these interactions before turning to the implementation. In the cases when  $c$  inhibits the interaction (activation/inhibition) between  $a$  and  $b$ , we want to disregard the interaction between  $a$  and  $b$  when  $c$  is active, and consider it as usual when  $c$  is inactive.

In the cases when  $c$  activates the interaction (activation/inhibition) between  $a$  and  $b$ , the situation is slightly different. In the semantics defined by our framework so far the interactions are ‘on’ by default, i.e., the potential effect of every interaction is considered when computing the next state of the system (of course if it is not inhibited by another protein as in the case above). However, this assumption does not allow us to model interactions that are active only under certain conditions, i.e., interactions that are ‘off’ by default. The notion

of activating an interaction is aimed at overcoming this issue. When we say that  $c$  activates the interaction between  $a$  and  $b$  we assume that this interaction is not functional without  $c$ . Thus, according to this reasoning the interaction between  $a$  and  $b$  is taken into account only when  $c$  is active.

#### 6.4.2.2 Adjustments to the S-rules

To accommodate interactions that target other interactions, we need to slightly change the perspective that we took while building our framework in Section 6.4.1. Initially, we assumed that every interaction in the network is present by default, which was reflected in the way we defined the interactions in a network, for example

$$activates(a, b, T).$$

By placing the time variable into the interaction definition we expressed that the interaction between  $a$  and  $b$  holds at any given time point. However, with the possibility to switch interactions on and off no longer this assumption holds and we need to change our formulation as follows

$$activates(a, b).$$

This we describes network structure independently of time, and the current state of the network decides whether an interaction is allowed at the current time point. Now, for example, the network depicted in Figure 6.4.3a can be expressed as follows

$$\begin{aligned} S1 : & \text{protein}(a). \\ S2 : & \text{protein}(b). \\ S3 : & \text{protein}(c). \\ S4 : & \text{activates}(a, b). \\ S5 : & \text{activates}(c, \text{activates}(a, b)). \end{aligned}$$

Further, we introduce a new type ‘interaction’ as follows

$$\begin{aligned} G14 : & \text{interaction}(\text{activates}(X, Y)) \leftarrow \text{activates}(X, Y). \\ G15 : & \text{interaction}(\text{inhibits}(X, Y)) \leftarrow \text{inhibits}(X, Y). \end{aligned}$$

These rules declare that predicates  $activates/2$  and  $inhibits/2$  are of type ‘interaction’. We need this type because now we can have an interaction as an argument of another interaction, so we need the means to distinguish between

proteins and interactions. Additionally, we introduce a general supertype ‘entity’ as follows

$$\begin{aligned} G14 : \text{entity}(X) &\leftarrow \text{protein}(X). \\ G15 : \text{entity}(X) &\leftarrow \text{interaction}(X). \end{aligned}$$

These rules say that both proteins and interactions are of type *entity*.

#### 6.4.2.3 Adjustments to the G-rules

Next, we need to adjust rules G8 and G9 to the new semantics. These rules, which count the number of activation and inhibition links with an active source in the current framework, rely on the fact that all interactions are allowed at any time step. To accommodate the new semantics we need to account for the fact that some interactions can be inhibited at one time step and active at another. These are the only G-rules that need to be modified. To support modified rules G8<sup>1</sup> and G9<sup>1</sup>, we add several extra rules as described below.

Firstly, we reformulate rules G8 and G9 as follows

$$\begin{aligned} G8^1 : \#\_act(Y, A - I, T) &\leftarrow \#\_potential\_act(Y, A), \\ &\quad I\{\text{inh\_act}(X, Y, T) : \text{activates}(X, Y) : \\ &\quad \text{protein}(X)\}I, \\ &\quad \text{protein}(Y), \text{int}(A), \text{int}(I). \\ G9^1 : \#\_inh(Y, A - I, T) &\leftarrow \#\_potential\_inh(Y, A), \\ &\quad I\{\text{inh\_inh}(X, Y, T) : \text{inhibits}(X, Y) : \\ &\quad \text{protein}(X)\}I, \\ &\quad \text{protein}(Y), \text{int}(A), \text{int}(I). \end{aligned}$$

where  $\#\_potential\_act(Y, A)$  and  $\#\_potential\_inh(Y, A)$  are used to calculate the number of activation/inhibition links that may potentially influence the node  $Y$  based on the network structure. These predicates are defined as follows

$$\begin{aligned} G8.1 : \#\_potential\_act(Y, A) &\leftarrow A\{\text{activates}(X, Y) : \text{protein}(X)\}A, \\ &\quad \text{protein}(Y), \text{int}(A). \\ G9.1 : \#\_potential\_inh(Y, I) &\leftarrow I\{\text{inhibits}(X, Y) : \text{protein}(X)\}I, \\ &\quad \text{protein}(Y), \text{int}(I). \end{aligned}$$

Rules G8.1 and G9.1 count the number of inbound activation/inhibition links without taking into account whether the interaction agent is active or

not. Rules G8<sup>1</sup> and G9<sup>1</sup> take this number and subtract from it the number of links that are inactive at the current time point due to the specific network configuration. This is implemented by means of the predicates *inh\_act*(*X*, *Y*, *T*) ( *inh\_inh*(*X*, *Y*, *T*) ) that reflect the fact that activation (inhibition) between *X* and *Y* at time point *T* is inhibited. According to the semantics defined in Section 6.4.2.1 an interaction can be inhibited in three cases.

The first case is derived from the previous framework: an interaction is not active if its trigger is not active. This condition can be implemented as follows

$$\begin{aligned} \text{G8.2 : } \text{inh\_act}(X, Y, T) &\leftarrow \text{not act}(X, T), \text{activates}(X, Y), \\ &\quad \text{protein}(X), \text{entity}(Y). \\ \text{G9.2 : } \text{inh\_inh}(X, Y, T) &\leftarrow \text{not act}(X, T), \text{inhibits}(X, Y), \\ &\quad \text{protein}(X), \text{entity}(Y). \end{aligned}$$

In rule G8.2 (G9.2) we say that if protein *X* potentially activates (inhibits) a protein or another interaction *Y*, but *X* is not active itself, then the interaction between *X* and *Y* (where *Y* may be an interaction itself) will be treated as inhibited. This is consistent with Section 6.4.1, where we do not count the influence of such an interaction on *Y* if *X* is not active, with the only difference that *Y* can be not only a protein but also an interaction. In this case, we do not need to check whether the interaction between *X* and *Y* is affected by other proteins (as in Figure 6.4.3), because without the active trigger this interaction will not fire anyway.

The second case when an interaction can be inhibited is when there exists an inhibition link that suppresses this interaction as in Figure 6.4.3b,d. This influence can be expressed as follows

$$\begin{aligned} \text{G8.3 : } \text{inh\_act}(X, Y, T) &\leftarrow \text{act}(Z, T), \text{inhibits}(Z, \text{activates}(X, Y)), \\ &\quad \text{protein}(Z), \text{protein}(X), \text{entity}(Y). \\ \text{G9.3 : } \text{inh\_inh}(X, Y, T) &\leftarrow \text{act}(Z, T), \text{inhibits}(Z, \text{inhibits}(X, Y)), \\ &\quad \text{protein}(Z), \text{protein}(X), \text{entity}(Y). \end{aligned}$$

Rule G8.3 says that if protein *X* potentially activates a protein or another interaction *Y*, but at the same time there exists an active protein *Z* that inhibits this activation, then the interaction between *X* and *Y* will be deactivated. The situation can be illustrated with Figure 6.4.3b where *c* is active. Rule G9.3 expresses the same idea for inhibition. However, note that the inhibiting link from *c* can be in turn deactivated as in Figure 6.4.4, which is taken into account in the following rule modification

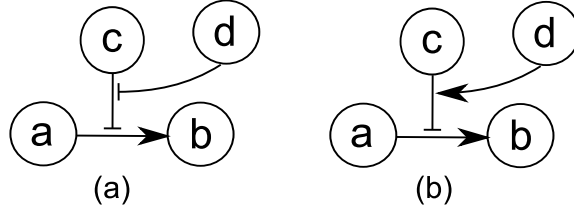


Figure 6.4.4: Examples of suppressed interactions. In (a) the inhibiting interaction from  $c$  will be deactivated if  $d$  is active, while in (b) it will be deactivated if  $d$  is not active

$$\begin{aligned}
 G8.3^1 : \text{inh\_act}(X, Y, T) &\leftarrow \text{act}(Z, T), \text{inhibits}(Z, \text{activates}(X, Y)), \\
 &\quad \text{not inh\_inh}(Z, \text{activates}(X, Y), T), \\
 &\quad \text{protein}(Z), \text{protein}(X), \text{entity}(Y). \\
 G9.3^1 : \text{inh\_inh}(X, Y, T) &\leftarrow \text{act}(Z, T), \text{inhibits}(Z, \text{inhibits}(X, Y)), \\
 &\quad \text{not inh\_inh}(Z, \text{inhibits}(X, Y), T), \\
 &\quad \text{protein}(Z), \text{protein}(X), \text{entity}(Y).
 \end{aligned}$$

Essentially, we just add the (recursive) condition that the link from  $Z$  to  $\text{activates}(X, Y)$  or  $\text{inhibits}(X, Y)$  should not be deactivated itself.

The third case when an interaction can be deactivated according to our semantics is when this interaction should be activated by an external protein, but this protein is not active. For example, in Figure 6.4.3a,c, the interaction between  $a$  and  $b$  will be inactive if  $c$  is not active. This condition can be captured by means of the following rules

$$\begin{aligned}
 G8.4 : \text{inh\_act}(X, Y, T) &\leftarrow \text{activates}(Z, \text{activates}(X, Y)), \\
 &\quad \text{inh\_act}(Z, \text{activates}(X, Y), T), \\
 &\quad \text{protein}(Z), \text{protein}(X), \text{entity}(Y). \\
 G9.4 : \text{inh\_inh}(X, Y, T) &\leftarrow \text{activates}(Z, \text{inhibits}(X, Y)), \\
 &\quad \text{inh\_act}(Z, \text{inhibits}(X, Y), T), \\
 &\quad \text{protein}(Z), \text{protein}(X), \text{entity}(Y).
 \end{aligned}$$

Rule G8.4 makes use of the previously defined rules: G8.2 and G8.3<sup>1</sup>. The interaction between  $X$  and  $Y$  will be considered inactive if  $Z$  is not active, or,

even if  $Z$  is active, if there is some other interaction that restricts the influence of  $Z$ .

The extended framework subsumes the one designed in Section 6.4.1, which means that the models built in that framework remain valid in the new framework with only a few alterations. Time argument should be removed from S-rules and every interaction should be typed with the *interaction* predicate.

**Example 11** Consider the network depicted in Figure 6.4.4a. To model this network we build the ASP framework with rules G1, G2<sup>3</sup>, G3<sup>3</sup>, G4, G5, G6, G7, G8<sup>1</sup>, G8.1, G8.2, G8.3<sup>1</sup>, G8.4, G9<sup>1</sup>, G9.1, G9.2, G9.3<sup>1</sup>, G9.4, G10, G10.1, G11, G11.1, G14, G15, G16, G17 and describe the network with the following rules

```

S1 : protein(a).
S2 : protein(b).
S2 : protein(c).
S2 : protein(d).
S3 : activates(a, b).
S4 : inhibits(c, activates(a, b)).
S5 : inhibits(d, inhibits(c, activates(a, b))).
S6 : act(a, 0).
S7 : inh(b, 0).
S8 : act(c, 0).
S9 : act(d, 0).

```

In this example, activation of  $a$  by  $b$  should be suppressed by  $c$ , because it is active. However,  $d$  is active as well, and it blocks the influence of  $c$ , thus eventually  $a$  will activate  $b$ . The answer set of this program is  $\{act(a, 0), inh(b, 0), act(c, 0), act(d, 0), act(a, 1), act(b, 1), act(c, 1), act(d, 1), act(a, 2), act(b, 2), act(c, 2), act(d, 2), act(a, 3), act(b, 3), act(c, 3), act(d, 3)\}$ . The steady state is  $\{act(a), act(b), act(c), act(d)\}$ .

#### 6.4.2.4 Extended framework summary

Here we summarize the modifications that we have introduced in Section 6.4.2 and present a set of rules that constitute the final version of our modelling

framework.

$G1 :$	$time(0..2).$	
$G2^3 :$	$act(Y, T)$	$\leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1),$ $act\_th(Y, Th), not\ ab(Y, T - 1),$ $A - I > Th, T > 0, int(A), int(I).$
$G3^3 :$	$inh(Y, T)$	$\leftarrow \#\_act(Y, A, T - 1), \#\_inh(Y, I, T - 1),$ $inh\_th(Y, Th), not\ ab(Y, T - 1),$ $I - A > Th, T > 0, int(A), int(I).$
$G4 :$		$\leftarrow act(X, T), inh(X, T).$
$G5 :$	$act(X, T)$	$\leftarrow act(X, T - 1), not\ inh(X, T), T > 0.$
$G6 :$	$inh(X, T)$	$\leftarrow inh(X, T - 1), not\ act(X, T), T > 0.$
$G7 :$	$int(0.. \#\_of\_prot).$	
$G8^1 :$	$\#\_act(Y, A - I, T)$	$\leftarrow \#\_potential\_act(Y, A),$ $I\{inh\_act(X, Y, T) : activates(X, Y) :$ $protein(X)\}I,$ $protein(Y), int(A), int(I).$
$G8.1 :$	$\#\_potential\_act(Y, A)$	$\leftarrow A\{activates(X, Y) : protein(X)\}A,$ $protein(Y), int(A).$
$G8.2 :$	$inh\_act(X, Y, T)$	$\leftarrow not\ act(X, T), activates(X, Y),$ $protein(X), entity(Y).$
$G8.3^1 :$	$inh\_act(X, Y, T)$	$\leftarrow act(Z, T), inhibits(Z, activates(X, Y)),$ $not\ inh\_inh(Z, activates(X, Y), T),$ $protein(Z), protein(X), entity(Y).$
$G8.4 :$	$inh\_act(X, Y, T)$	$\leftarrow activates(Z, activates(X, Y)),$ $inh\_act(Z, activates(X, Y), T),$ $protein(Z), protein(X), entity(Y).$
$G9^1 :$	$\#\_inh(Y, A - I, T)$	$\leftarrow \#\_potential\_inh(Y, A),$ $I\{inh\_inh(X, Y, T) : inhibits(X, Y) :$ $protein(X)\}I,$ $protein(Y), int(A), int(I).$
$G9.1 :$	$\#\_potential\_inh(Y, I)$	$\leftarrow I\{inhibits(X, Y) : protein(X)\}I,$ $protein(Y), int(I).$
$G9.2 :$	$inh\_inh(X, Y, T)$	$\leftarrow not\ act(X, T), inhibits(X, Y),$ $protein(X), entity(Y).$
$G9.3^1 :$	$inh\_inh(X, Y, T)$	$\leftarrow act(Z, T), inhibits(Z, inhibits(X, Y)),$ $not\ inh\_inh(Z, inhibits(X, Y), T),$ $protein(Z), protein(X), entity(Y).$



$G9.4 :$	$inh\_inh(X, Y, T)$	$\leftarrow$	$activates(Z, inhibits(X, Y)),$ $inh\_act(Z, inhibits(X, Y), T),$ $protein(Z), protein(X), entity(Y).$
$G10 :$	$act\_th(X, 0)$	$\leftarrow$	$not\ mod\_act\_th(X).$
$G10.1 :$	$mod\_act\_th(X)$	$\leftarrow$	$act\_th(X, Th), Th \neq 0.$
$G11 :$	$inh\_th(X, 0)$	$\leftarrow$	$not\ mod\_inh\_th(X).$
$G11.1 :$	$mod\_inh\_th(X)$	$\leftarrow$	$inh\_th(X, Th), Th \neq 0.$
$G12 :$	$act(X, 0)$	$\leftarrow$	$not\ inh(X, 0).$
$G13 :$	$inh(X, 0)$	$\leftarrow$	$not\ act(X, 0).$
$G14 :$	$interaction(activates(X, Y))$	$\leftarrow$	$activates(X, Y).$
$G15 :$	$interaction(inhibits(X, Y))$	$\leftarrow$	$inhibits(X, Y).$

## 6.5 Network modelling and analysis algorithms

Each network has  $2^N$  possible states, where  $N$  is the number of proteins in the network. As shown in Example 2, in the answer set representation, the state of a protein corresponds to a predicate in the set, and a network state corresponds to a subset that contains all protein states for one time point. For example,  $\{act(a, 1), act(b, 1)\}$  is a part of the answer set for the program in Example 2 that describes the state of the network at time point 1. There are two problems associated with finding the steady states of a network by solving a corresponding answer set program as proposed in Section 6.4. First of all, when computing a trajectory from an initial state it is impossible to estimate how many time steps are needed (the upper time bound). Trajectories within too short time intervals may not reach the steady state, while too long intervals increase computational expenses. For instance, if we limit the upper time bound in Example 2 by 1 (set rule  $G1$  as  $time(0..1).$ ) the steady state will not be present in the answer set. On the other hand, if we set the upper time bound to e.g. 5, we will find a steady state but at the same time we will compute 2 extra states that do not contain any additional information about the network behaviour.

Secondly, the problem which follows from the first one, is related to the attraction basin computation. An attraction basin is a set of trajectories that converge to the same steady state (see Section 4.3 for details). Finding these trajectories is computationally expensive, as we need to consider the whole network state space. We can iterate over all possible initial states, but for every initial state we need to adjust the time interval every time, which, again, makes the whole process computationally heavy.

This problem was addressed in Chapter 5, where we have proposed the Temporal Algorithm to efficiently solve Markovian programs. As it can be seen from Section 6.4, answer set programs that built in our ASP framework conform to the definition of Markovian programs, and thus the Temporal Algorithm can be applied to find the steady states of these networks.

Moreover, our framework, besides being Markovian, exhibits other properties that allow for introduction of another algorithm for steady states computation as explained in Section 6.5.1. A new algorithm called State Space Building (SSB) algorithm, can be used to build a state space of a network described in our framework and to find the steady states occurring in this space. This algorithm allows building a state space in one ASP solver run, which makes it superior to the Temporal algorithm for the task of state space building. However, more fine grained analysis such as finding particular trajectories and querying a state space in general is not straightforward with the SSB approach. Therefore, in Section 6.5.2 we propose an extension of the Temporal algorithm that makes possible to pose arbitrary queries about a network state space.

### 6.5.1 State Space Building approach

To explain an efficient way of finding the steady states of a network, let us point out the following fact: any answer set obtained with the framework we proposed contains a complete description of the network state at any time point, i.e., for every node  $n$  either fact  $act(n, T)$  or  $inh(n, T)$  is defined. This trivially follows from the framework definition, where the state for every node is defined by the states of other nodes or by the inertia rules G5 and G6. Predicates  $act$  and  $inh$  define the state of the network, thus every answer set of a program in our framework will contain a complete description of the network state at every time point. This is an important observation, because partial network state definitions do not make sense in our framework.

The idea of the State Space Building (SSB) algorithm is based on two observations. The first observation is that since the complete network state is described explicitly by the predicates  $act$  and  $inh$ , we can automatically generate all possible states of a network by means of rules G12 and G13, explained in Section 6.4.1.6. The second observation is that our modelling framework is a Markovian program, which means that any next state depends only on its direct predecessor. Combining these observations together, we can build a framework with rules G1, G2<sup>3</sup>, G3<sup>3</sup>, G4, G5, G6, G7, G8<sup>1</sup>, G8.1, G8.2, G8.3<sup>1</sup>, G8.4, G9<sup>1</sup>, G9.1, G9.2, G9.3<sup>1</sup>, G9.4, G10, G10.1, G11, G11.1, G12, G13, G14, G15 and extend it with the description of a network presented as a set of S-rules (the

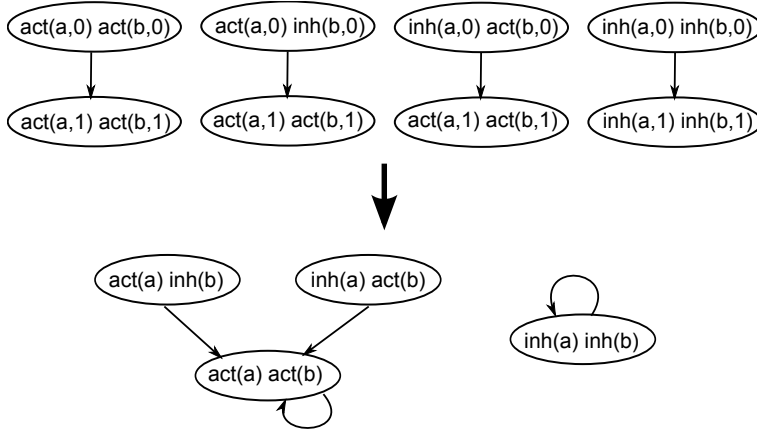


Figure 6.5.1: The SSB algorithm execution for the network depicted in Figure 6.3.1a. Nodes represent the states of the network. The topmost part of the figure denotes the transitions obtained by the answer set framework, while the bottom part denotes the state space of the network built with the SSB algorithm.

actual instances of S-rules are non-essential for the algorithm). Here, rules G12 and G13 provide all possible network states at time point 0. We explicitly fix the upper temporal bound by setting rule G1 to  $\text{time}(0..1)$ , which restricts the program to find only immediate consequent states for any given starting state. Now, solving this program will give us at least  $2^N$  answer sets, where each answer set will contain an initial state and a transition of this state to a possible following state, which in turn appears as initial state in some other answer set (due to the complete starting set enumeration). In this way we obtain a complete set of network transitions that can be merged in a network state space graph. The SSB algorithm execution for the network depicted in Figure 6.3.1a is shown in Figure 6.5.1.

Note that the SSB algorithm can be efficiently distributed in a multi-threaded environment, because every state transition can be computed independently. Indeed, if we consider every possible state of the network as a starting state, then the only operations we need are to compute the network evolution for one step, and then combine the obtained state-state transitions and join them together to obtain all possible trajectories.

### 6.5.2 Querying networks

The SSB algorithm described above is well suited to build the whole state space of a network, however its applicability for network analysis is limited. Consider the case when we want to find only trajectories that conform to certain constraints. For example, in a network that contains proteins  $a$ ,  $b$  and  $c$  we want to get only the trajectories where  $b$  is active only after  $a$  is inhibited, independently of the state of  $c$ . This is not possible to do with SSB unless we build the whole state space first. However, this can be a computationally expensive operation, as the number of states of a network grows exponentially with the number of nodes. Thus, we need a more efficient way to deal with the task of answering queries about network states and trajectories, which is described below.

The representation of regulatory networks with ASP programs makes such reasoning about network behaviour much easier than in the boolean setting. In this section we explain how we can plug in queries for a (partial) network state.

Let us consider the situation that a biologist observes a certain state of a regulatory network and would like to get an explanation of how this state can be reached in the model, i.e., the trajectories that lead to this state. Let us consider the example where a biologist wants to get the trajectories, which lead to states where  $a$  is active and  $b$  is inhibited. Such a query can be incorporated by adding to a network description the following set of rules

$$\begin{aligned} q &\leftarrow act(a, T), inh(b, T), time(T). \\ &\leftarrow not\ q. \end{aligned}$$

Intuitively, the first rule says that query  $q$  would appear in an answer set when  $a$  is active and  $b$  is inhibited. The second rule is a constraint which says that any answer set of our program should contain  $q$ , i.e., the query should be satisfied. However, just plugging these lines in our program would not lead to the desired result. First of all, the first rule is not valid in terms of Markovian programs, as its body has time-dependent predicates while the head is not time-dependent. This can be fixed by replacing  $q$  with  $q(T)$ . However, the second problem is more serious - as it was explained in Section 5.5 we compute the states of a network for each time step separately, and the query would be added to the program for every time step according to the definition of partial grounding. This would lead to the fact that only states that satisfy the query would be in answer sets, and the actual trajectories would not be computed, which is not the desired behaviour.

To overcome this problem, we make a straightforward extension of the Temporal algorithm that handles a query as a separate program. As described in

Section 5.5 the algorithm keeps all possible trajectories that have been reached at the current time point. As every trajectory is represented as a sequence of states, we can impose our queries on this sequence.

**Require:**  $\mathbf{P} = \langle P, \tau \rangle$ ; {A Markovian program}

**Require:**  $Q$ ; {Query program}

```

1:  $\mathbf{V} = \emptyset$ ; {The set of visited states}
2:  $\mathcal{A}_{-1} = \text{List}(\text{AS}(P^e))$ ; {The list of sets of initial conditions that will be used
   for grounding}
3:  $n = \text{size}(\mathcal{A}_{-1})$ ;
4:  $\mathcal{A} = \text{List}(\mathcal{A}_{-1}[k] | 1 \leq k \leq n)$ ; {The list of sets of current states that are
   rooted in the  $k$ -th set of initial conditions}
5:  $\mathcal{T} = \mathcal{A}_{-1}$ ; {The list of trajectories}
6:  $t = 0$ ;
7: repeat
8:   Obtain the partial temporal grounding  $P_t$  of  $\mathbf{P}$ ;
9:   for  $k = 1$  to  $n$  do
10:     $A^{-1} = \mathcal{A}_{-1}[k]$ ; {Obtain the  $k$ -th set of initial conditions}
11:     $\mathbf{A}_{\text{new}} = \emptyset$ ;
12:    {Iterate over the sets of current states rooted in the  $k$ -th set of initial
     conditions}
13:    for all  $\mathbf{A} \in \mathcal{A}[k]$  do
14:      {Iterate over the current states rooted in the  $k$ -th set of initial condi-
       tions}
15:      for all  $A^{t-1} \in \mathbf{A}$  do
16:         $P'_t = \text{Gnd}(P_t \cup \{l \leftarrow . | l \in A^{t-1} \cup A^{-1}\})$ ; {Grounding  $P_t$ }
17:         $S = \text{AS}(R^{A^{t-1} \cup A^{-1}}(P'_t))$ ; {Obtaining states for time point  $t$ }
18:        for all  $A^t \in S$  do
19:          Add state  $A^t$  to all trajectories in  $\mathcal{T}$  that contain  $A^{t-1}$ 
20:          query_satisfied = true;
21:          for all trajectories  $T$  that end with  $A^t$  do
22:            if  $\text{AS}(Q \cup \{t \leftarrow . | t \in T\} \cup \{\text{time}(0..t).\}) = \emptyset$  then
23:              query_satisfied = false;
24:            else
25:               $\mathcal{T}' = \mathcal{T}' \cup T$ ;
26:            end if
27:          end for
28:          if  $A^t$  is not in  $\mathbf{V}$  then
29:            Add state  $A^t$  to the set  $\mathbf{V}$  of visited states

```

```

30:         if query_satisfied=false then
31:             Add state  $A^t$  to the set  $\mathbf{A}_{new}$  of states that should be con-
               sidered in the following iteration
32:         end if
33:     end if
34: end for
35: end for
36: end for
37:  $\mathcal{A}[k] = \mathbf{A}_{new};$ 
38: end for
39:  $t = t + 1;$ 
40: until  $\mathcal{A}$  is empty or  $\mathcal{A}$  contains only empty sets
41: return  $\mathcal{T}$ 

```

The new lines added to the original Temporal algorithm are 20-27, 30 and 32. The effect of these lines is that when extending trajectories with newly computed states, we check whether a trajectory satisfies the query  $Q$ . If the trajectory satisfies  $Q$ , we add it to the resulting set and do not consider this trajectory in the following steps.

**Example 12** Let us consider a program in our framework built for the network in Figure 6.3.1b and suppose that we want to know the trajectories that lead to the states where  $a$  is inhibited. The query program would look as follows

$$\begin{aligned}
 q &\leftarrow inh(a, T), time(T). \\
 &\leftarrow not\ q.
 \end{aligned}$$

The Temporal algorithm computes all possible states for time point 0:  $\{act(a, 0), inh(b, 0)\}, \{inh(a, 0), inh(b, 0)\}, \{act(a, 0), act(b, 0)\}, \{inh(a, 0), act(b, 0)\}$ . These answer sets lead to four query evaluations, for example for the first state the query program would be

$$\begin{aligned}
 &time(0). \\
 &act(a, 0). \quad \quad \quad inh(b, 0). \\
 q &\leftarrow inh(a, 0), time(0). \\
 &\leftarrow not\ q.
 \end{aligned}$$

If a query program has an answer set, this means that the query is satisfied and the trajectory that leads to this query is marked and not considered on the following steps of the Temporal algorithm.

After time point 0, states  $\{inh(a, 0), act(b, 0)\}$  and  $\{inh(a, 0), inh(b, 0)\}$  satisfy the query and are not considered further. The state  $\{act(a, 0), act(b, 0)\}$  leads to the state  $\{inh(a, 1), act(b, 1)\}$ , and the query program for this trajectory is

$time(0).$	$time(1).$
$act(a, 0).$	$act(b, 0).$
$inh(a, 1).$	$act(b, 1).$
$q$	$\leftarrow inh(a, 0).$
$q$	$\leftarrow inh(a, 1).$
	$\leftarrow not\ q.$

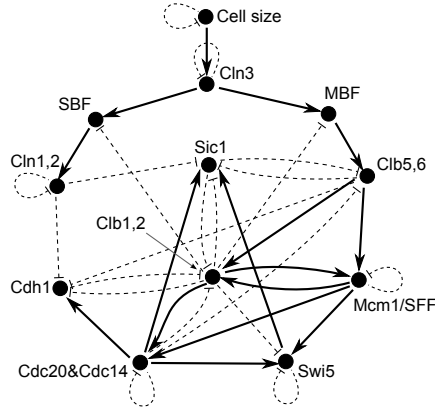
The program has an answer set, thus the query is satisfied and the trajectory is marked and not considered for further processing. The algorithm proceeds until it finds all steady states to guarantee that no trajectory satisfying the query is omitted, and then all marked trajectories are returned to the user. After the Temporal algorithm finishes, it outputs four trajectories that lead to the query state:

1.  $[\{inh(a, 0), act(b, 0)\}]$
2.  $[\{inh(a, 0), inh(b, 0)\}]$
3.  $[\{act(a, 0), act(b, 0)\}, \{inh(a, 1), act(b, 1)\}]$
4.  $[\{act(a, 0), inh(b, 0)\}, \{act(a, 1), act(b, 1)\}, \{inh(a, 2), act(b, 2)\}]$

Note, that a query can be incomplete, i.e., we are not forced to specify the complete state of the network, as shown in Example 12. Another appealing feature of the query mechanism is that the whole trajectory is exposed to the query program. This means that arbitrary conditions not only on the current state but also on any previous states can be imposed to a query, as illustrated further in Example 13. Note that this example does not have to conform to the Markovian program definition since query program is an extra component, not a part of the modelling framework.

**Example 13** Since queries are ASP programs themselves, we can construct composite queries with minimal effort. Consider a network with nodes  $a$ ,  $b$  and  $c$ . A simple program

$$\begin{aligned} q &\leftarrow inh(a, T), act(b, T_1), T < T_1. \\ &\leftarrow not\ q. \end{aligned}$$



## 6.6 Experimental validation

In this section we describe three proof-of-concept experiments for the analysis of real interaction networks. As Boolean networks are a popular formalism to describe cell cycle processes, we selected three cell cycle networks represented with boolean models and model them in our framework. First, we take the dynamical network model of the *Budding Yeast* cell cycle network [94]. The second experiment involves the modelling of the *Fission Yeast* cell cycle network [35]. These two experiments are performed within the basic framework, described in Section 6.4.1, as they do not require the extended expressivity of the full framework. The third experiment was performed with the *Mammalian* cell cycle network, which was modelled with the full framework described in Section 6.4.2. We use the **clingo** [61] system that combines a grounder and solver



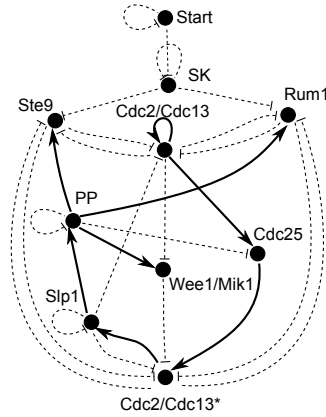


Figure 6.6.2: The dynamical network model of the *Fission Yeast* cell cycle [35].

in one package to compute the answer sets.

### 6.6.1 Budding Yeast network

The structure of the Budding Yeast network, depicted in Figure 6.6.1, is captured by means of the following S-rules:

```

protein(cln3).    protein(clb56).
protein(sbf).    protein(cdh1).
protein(mbf).    protein(clb12).
protein(cln12).  protein(mcm1).
protein(sic1).   protein(cdc20).
protein(swi5).
```

<i>inhibits(cln3, cln3, T).</i>	<i>activates(cln3, mbf, T).</i>
<i>inhibits(cdc20, cdc20, T).</i>	<i>activates(mcm1, cdc20, T).</i>
<i>inhibits(cln12, cln12, T).</i>	<i>activates(cln3, sbf, T).</i>
<i>inhibits(mcm1, mcm1, T).</i>	<i>activates(swi5, sic1, T).</i>
<i>inhibits(sic1, clb56, T).</i>	<i>activates(mbf, clb56, T).</i>
<i>inhibits(clb12, swi5, T).</i>	<i>activates(cdc20, swi5, T).</i>
<i>inhibits(clb56, sic1, T).</i>	<i>activates(clb56, mcm1, T).</i>
<i>inhibits(clb12, cdh1, T).</i>	<i>activates(cdc20, sic1, T).</i>
<i>inhibits(sic1, clb12, T).</i>	<i>activates(clb56, clb12, T).</i>
<i>inhibits(cdh1, clb12, T).</i>	<i>activates(cdc20, cdh1, T).</i>
<i>inhibits(clb12, sic1, T).</i>	<i>activates(mcm1, clb12, T).</i>
<i>inhibits(cdc20, clb12, T).</i>	<i>activates(clb12, cdc20, T).</i>
<i>inhibits(cln12, sic1, T).</i>	<i>activates(clb12, mcm1, T).</i>
<i>inhibits(cdc20, clb56, T).</i>	<i>activates(sbf, cln12, T).</i>
<i>inhibits(clb12, sbf, T).</i>	<i>activates(mcm1, swi5, T).</i>
<i>inhibits(clb56, cdh1, T).</i>	
<i>inhibits(clb12, mbf, T).</i>	
<i>inhibits(cln12, cdh1, T).</i>	
<i>inhibits(swi5, swi5, T).</i>	

There are 11 proteins, 19 inhibiting and 15 activating interactions in total, each modelled with an S-rule.

time	cln3	mbf	sbf	cln12	cdh1	swi5	cdc20	clb56	sic1	clb12	mcm1
1	1	0	0	0	1	0	0	0	1	0	0
2	0	1	1	0	1	0	0	0	1	0	0
3	0	1	1	1	1	0	0	0	1	0	0
4	0	1	1	1	0	0	0	0	0	0	0
5	0	1	1	1	0	0	0	1	0	0	0
6	0	1	1	1	0	0	0	1	0	1	1
7	0	0	0	1	0	0	1	1	0	1	1
8	0	0	0	0	0	1	1	0	0	1	1
9	0	0	0	0	0	1	1	0	1	1	1
10	0	0	0	0	0	1	1	0	1	0	1
11	0	0	0	0	1	1	1	0	1	0	0
12	0	0	0	0	1	1	0	0	1	0	0
13	0	0	0	0	1	0	0	0	1	0	0
14	0	0	0	0	1	0	0	0	1	0	0

Table 6.2: Budding Yeast network execution flow

In Table 6.2 we provide the results for the Budding Yeast network trajectories respectively from an initial state that corresponds to the point of cell division initiation. The table is a structured representation of the derived answer set. Rows stand for time steps, and columns stand for proteins, i.e. one column shows the changes of one particular protein over time. For brevity, we denote an *act*/2 predicate with 1 in the appropriate cell and *inh*/2 with 0. For example, *act*(*SK*, 1) converts to 1 in row 1 and column *SK*, etc. The same description applied to Tables 6.3 and 6.4 that presented further.

Table 6.2 corresponds to the model of cell division process occurring in the Budding Yeast. It starts with an initial state in which the cell have enough resources to start the division process and eventually the model arrives at a steady state as can be seen in rows 13 and 14, which corresponds to a steady state of the cell. This trajectory and the resulting steady state have also been found by Li et al. [94]. All other steady states found by our framework have also been reported in the work [94], which proves that the models exhibit identical behaviour. In Figure 6.6.3 the largest attraction basin of this network is shown.

## 6.6.2 Fission Yeast network

The structure of the Fission Yeast network, depicted in Figure 6.6.2, is captured by means of the following S-rules

<i>protein</i> ( <i>sk</i> ).	<i>protein</i> ( <i>cdc25</i> ).
<i>protein</i> ( <i>ste9</i> ).	<i>protein</i> ( <i>pp</i> ).
<i>protein</i> ( <i>rum1</i> ).	<i>protein</i> ( <i>wee1</i> ).
<i>protein</i> ( <i>cdc2</i> ).	<i>protein</i> ( <i>slp1</i> ).
<i>protein</i> ( <i>cdc2_star</i> ).	<i>protein</i> ( <i>start</i> ).
<i>inhibits</i> ( <i>sk</i> , <i>ste9</i> , <i>T</i> ).	<i>inhibits</i> ( <i>cdc2</i> , <i>wee1</i> , <i>T</i> ).
<i>inhibits</i> ( <i>sk</i> , <i>rum1</i> , <i>T</i> ).	<i>inhibits</i> ( <i>ste9</i> , <i>cdc2_*</i> , <i>T</i> ).
<i>inhibits</i> ( <i>cdc2</i> , <i>ste9</i> , <i>T</i> ).	<i>inhibits</i> ( <i>slp1</i> , <i>cdc2</i> , <i>T</i> ).
<i>inhibits</i> ( <i>ste9</i> , <i>cdc2</i> , <i>T</i> ).	<i>inhibits</i> ( <i>cdc2_*</i> , <i>ste9</i> , <i>T</i> ).
<i>inhibits</i> ( <i>cdc2</i> , <i>rum1</i> , <i>T</i> ).	<i>inhibits</i> ( <i>rum1</i> , <i>cdc2_*</i> , <i>T</i> ).
<i>inhibits</i> ( <i>rum1</i> , <i>cdc2</i> , <i>T</i> ).	<i>inhibits</i> ( <i>cdc2_*</i> , <i>rum1</i> , <i>T</i> ).
<i>inhibits</i> ( <i>pp</i> , <i>cdc25</i> , <i>T</i> ).	<i>inhibits</i> ( <i>slp1</i> , <i>cdc2_*</i> , <i>T</i> ).
<i>inhibits</i> ( <i>pp</i> , <i>pp</i> , <i>T</i> ).	<i>inhibits</i> ( <i>slp1</i> , <i>slp1</i> , <i>T</i> ).
<i>inhibits</i> ( <i>sk</i> , <i>sk</i> , <i>T</i> ).	<i>inhibits</i> ( <i>start</i> , <i>start</i> , <i>T</i> ).
<i>inhibits</i> ( <i>wee1</i> , <i>cdc2_*</i> , <i>T</i> ).	

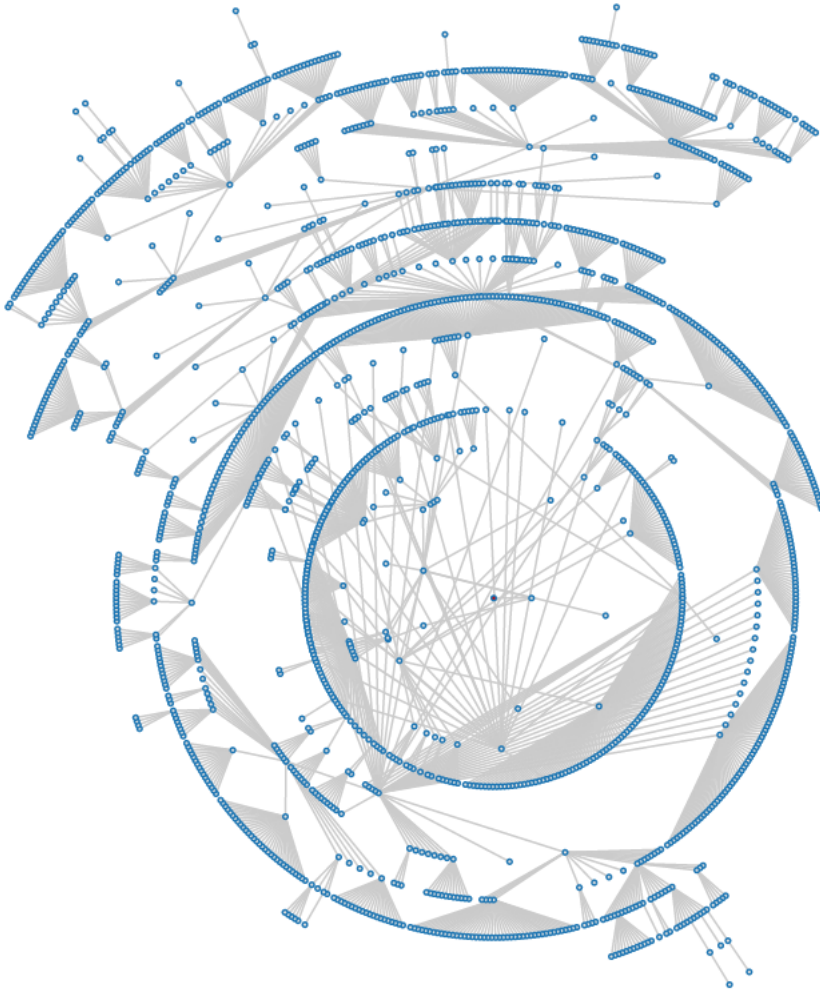


Figure 6.6.3: The biggest attraction basin of the Budding yeast cell cycle network model

```

activates(cdc2,cdc25,T).   activates(slp1,pp,T).
activates(pp,ste9,T).     activates(cdc25,cdc2_*,T).
activates(pp,rum1,T).     activates(cdc2_*,slp1,T).
activates(pp,wee1,T).     activates(start,sk).

inh_th(cdc2_*,-1).
act_th(cdc2,-1).

```

There are 10 proteins, 18 inhibiting and 7 activating interactions in total, each modelled with an S-rule.

Some facts such as the activation threshold of *Cdc2/Cdc13* or the inhibition threshold of *Cdc2/Cdc13\** are not explicitly seen on the network in Figure 6.6.2; to figure out the exact execution flow one has to read the corresponding article [35]. This shows one of the advantages of an ASP model in comparison with Boolean networks. The ASP model is self-descriptive, i.e., it does not rely on any implicit assumptions or background knowledge, while with Boolean networks, the user needs to check the conditions for every node before execution.

time	start	SK	cdc2	ste9	rum1	slp1	cdc2_*	wee1	cdc25	pp
1	1	0	0	1	1	0	0	1	0	0
2	0	1	0	1	1	0	0	1	0	0
3	0	0	0	0	0	0	0	1	0	0
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	0	0	0	0	1	0
6	0	0	1	0	0	0	1	0	1	0
7	0	0	1	0	0	1	1	0	1	0
8	0	0	0	0	0	1	0	0	1	1
9	0	0	0	1	1	0	0	1	0	1
10	0	0	0	1	1	0	0	1	0	0
11	0	0	0	1	1	0	0	1	0	0

Table 6.3: Fission Yeast network execution flow

Table 6.3 presents the division Process of the Fission Yeast, where the initial state of the network corresponds to a state of a cell ready for division and the states at time points 10 and 11 represent the steady state of the cell after the division finishes. This trajectory and the resulting steady state have also been found in the work of Davidich and Bornholdt [35]. All other steady states found by our framework have also been reported this work. In Figure 6.6.4 the largest attraction basin of this network is shown and in Figure 6.6.5 an interesting basin that attracts to a steady cycle is shown.

### 6.6.3 Mammalian cell cycle network

Faure et al. [49] constructed a boolean model of the mammalian cell cycle network. The original structure of this network is depicted in Figure 6.6.6. We argue that this version of the model suffers from the constraints imposed by the boolean network formalism, which in turn leads to some unintuitive modelling choices.

Let us first focus on the relationship between the cyclin-dependent kinase inhibitor *p27/Kip1* (denote as *p27* in the model) and *cdk2/Cyclin A* (denoted as *CycA*). When both *p27* and *CycA* are active, *p27* forms a complex with *CycA* and blocks activity of *CycA*. However, the cyclin remains present, and to model this fact rather than drawing an inhibiting edge from *p27* to *CycA* the blocking effect of *p27* is represented by drawing edges from *p27* to the targets of *CycA*, but with opposite signs, i.e. if *CycA* activates *E2F* then there is also an inhibiting edge between *p27* and *E2F*. Apparently, this is a workaround due to the fact that in Boolean networks it is not possible to declare an interaction between a node and an edge.

This problem arises in modelling the interactions of *p27* with *CycA* and *CycE* but also in modelling the activity of the Anaphase Promoting Complex (APC). APC is responsible for the progression and proper finishing of mitosis (the separation of the cell into two cells) and is presented in the model by proteins *Cdh1* and *Cdc20*. *Cdh1* is known to be an inhibitor of *CycA*; however, during the transition from the growth phase G2 to mitosis, the level of *CycA* is risen high enough to inactivate *Cdh1*. Since *Cdh1* is active in the G2 phase, it was long unclear how *CycA* can overrule the inhibitory influence of *Cdh1*. Recent research has revealed the role of the *E2 ubiquitin conjugating enzyme UbH10* in this process by showing that *Cdh1*-dependent degradation of *CycA* can happen only in the presence of *UbH10* [132]. Moreover, *Cdh1* triggers *UbH10* ubiquitination, but only in the case when the *Cdh1* targets are not active, or in the terms of the logical model, when *CycA*, *CycB* and *Cdc20* are inactive. These observations



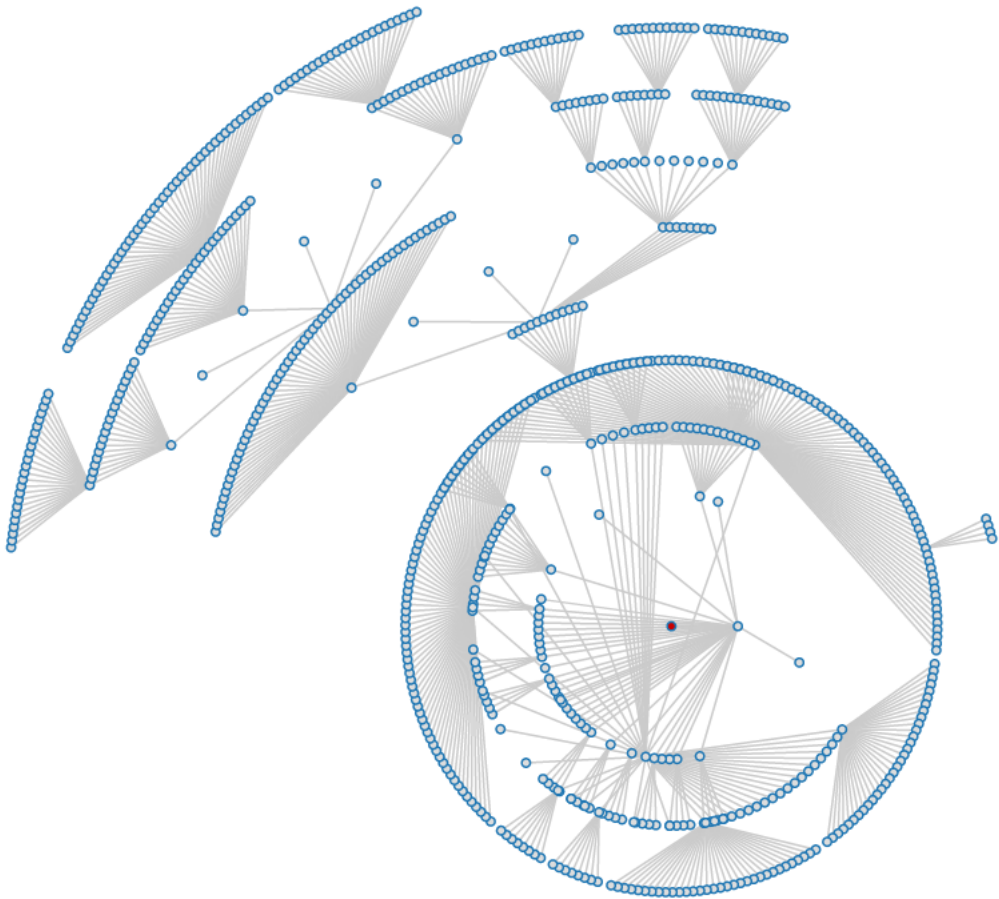


Figure 6.6.4: The biggest attraction basin of the Fission yeast cell cycle network model

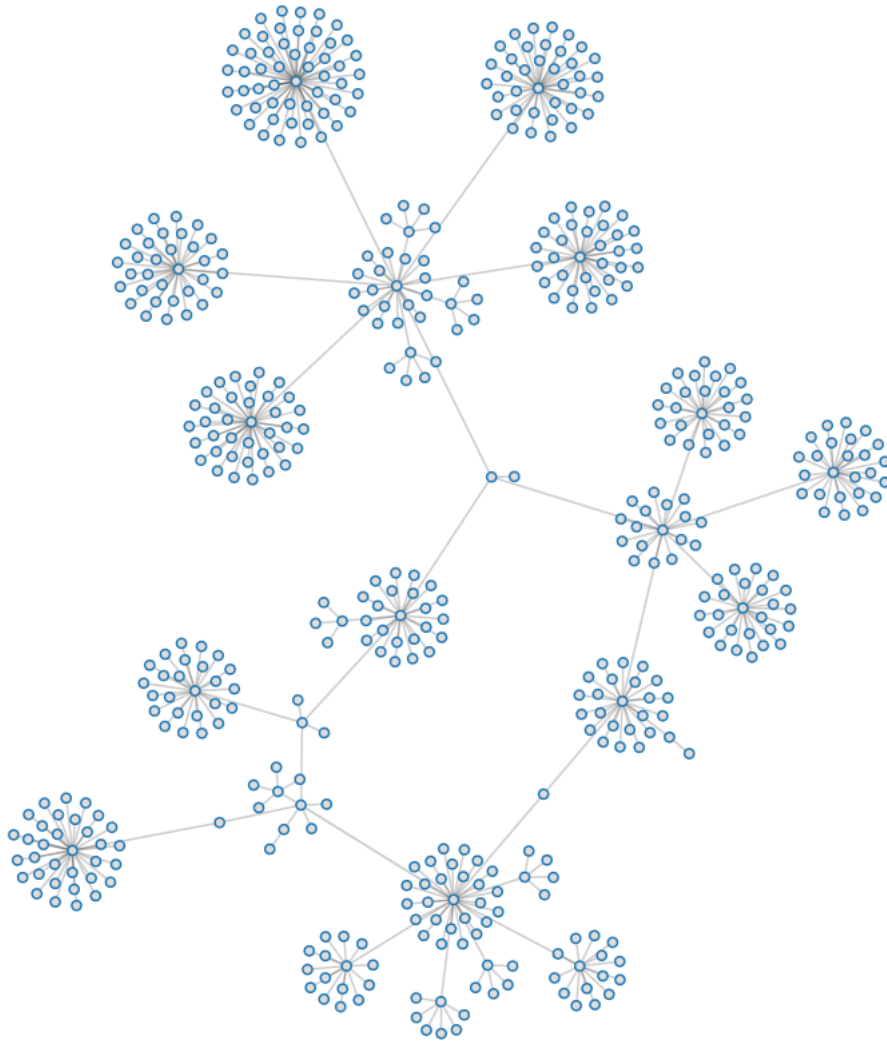


Figure 6.6.5: The attraction cycle of the Fission yeast cell cycle network model

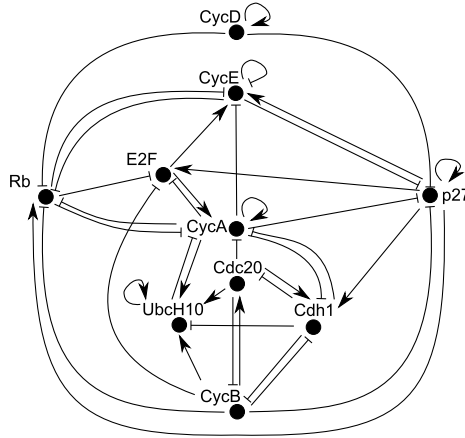


Figure 6.6.6: The model of the mammalian cell cycle regulatory network [49].

were formalized [49] and implicitly depicted in the network in Figure 6.6.6 by means of activation edges from the *Cdh1* targets to *UbcH10* and by an inhibition edge from *Cdh1* to *UbcH10*. Note again, that the relationships among the proteins in this model do not directly reflect the facts stated above.

After the establishment of the new framework we can address the aforementioned issues and create a more understandable model of the mammalian cell cycle shown in Figure 6.6.7. First of all, we can directly model the influence of *p27* on cyclin A and Cyclin E complexes. Now *p27* influences not the targets of these cyclin complexes, but their interactions with other entities directly using the following rules

```

inhibits(p27, inhibits(cycE, rb)).
inhibits(p27, inhibits(cycE, p27)).
inhibits(p27, inhibits(cycA, p27)).
inhibits(p27, inhibits(cycA, rb)).
inhibits(p27, inhibits(cycA, cdh1)).
inhibits(p27, inhibits(cycA, e2f)).
inhibits(p27, inhibits(cycA, cycE)).

```

Moreover, the functioning of APC can now be described more intuitively, as now we can model the fact that *cdh1* inhibits *CycA* only in the presence of

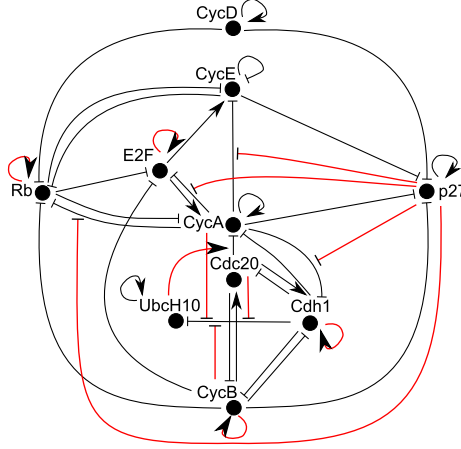


Figure 6.6.7: The mammalian cell cycle network model represented with the updated framework. Updated edges are shown in red.

*UbcH10* with the following rule

$$\text{activates}(\text{ubcH10}, \text{inhibits}(\text{cdh1}, \text{cycA})).$$

Additionally, we can model the fact that the APC substrates CycA, CycB and Cdc20 prevent *UbcH10* from degradation using the following rules

$$\begin{aligned} &\text{inhibits}(\text{cycB}, \text{inhibits}(\text{cdh1}, \text{ubcH10})). \\ &\text{inhibits}(\text{cdc20}, \text{inhibits}(\text{cdh1}, \text{ubcH10})). \\ &\text{inhibits}(\text{cycA}, \text{inhibits}(\text{cdh1}, \text{ubcH10})). \end{aligned}$$

In this way, we simplify the interaction diagram by redirecting some interaction edges, but also make it more intuitive as relationships such as ‘*p27* inhibits the activity of *Cyclin A complex*’ can be expressed in a more straightforward way.

Another peculiarity of the existing boolean model that we did not address yet is concerned with the inhibition of *p27*. According to the original model, *p27* is inhibited only when both *CycA* and *CycE* are active, which contradicts the boolean threshold semantics implemented in our framework. We can model this behaviour by exploiting the exception mechanism, thus representing this

condition as follows

$$\begin{aligned}
 ab(p27, T). & \\
 inh(p27, T) & \leftarrow cyclins\_active(cycA, cycE, T - 1), T > 0. \\
 act(p27, T) & \leftarrow not\ cyclins\_active(cycA, cycE, T - 1), \\
 & \quad active(p27, T - 1), T > 0. \\
 cyclins\_active(cycA, cycE, T) & \leftarrow act(cycA, T), act(cycE, T).
 \end{aligned}$$

which say that *p27* is inhibited if both *CycA* and *CycE* are active and it active otherwise. The final structure of the mammalian cell cycle network is captured by means of the following S-rules

<i>protein(cycA).</i>	<i>inhibits(cycA, inhibits(cdh1, ubc)).</i>
<i>protein(cycB).</i>	<i>inhibits(cycB, inhibits(cdh1, ubc)).</i>
<i>protein(cycD).</i>	<i>inhibits(cdc20, inhibits(cycA, cdh1)).</i>
<i>protein(cycE).</i>	<i>inhibits(cdc20, inhibits(cdh1, ubc)).</i>
<i>protein(rb).</i>	<i>activates(ubc, inhibits(cdh1, cycA)).</i>
<i>protein(e2f).</i>	<i>inhibits(p27, inhibits(cycE, rb)).</i>
<i>protein(p27).</i>	<i>inhibits(p27, inhibits(cycE, p27)).</i>
<i>protein(cdc20).</i>	<i>inhibits(p27, inhibits(cycA, p27)).</i>
<i>protein(cdh1).</i>	<i>inhibits(p27, inhibits(cycA, rb)).</i>
<i>protein(ubc).</i>	<i>inhibits(p27, inhibits(cycA, cdh1)).</i>
	<i>inhibits(p27, inhibits(cycA, e2f)).</i>
	<i>inhibits(p27, inhibits(cycA, cycE)).</i>
<i>inhibits(cycD, rb).</i>	<i>inhibits(cdh1, cycA).</i>
<i>inhibits(cycD, p27).</i>	<i>inhibits(cdh1, cdc20).</i>
<i>inhibits(rb, e2f).</i>	<i>inhibits(cdh1, ubc).</i>
<i>inhibits(cycE, p27).</i>	<i>inhibits(cdh1, cycB).</i>
<i>inhibits(cycE, rb).</i>	<i>inhibits(cycB, p27).</i>
<i>inhibits(cycA, rb).</i>	<i>inhibits(cycB, e2f).</i>
<i>inhibits(cycA, e2f).</i>	<i>inhibits(cdc20, cycB).</i>
<i>inhibits(cycA, cycE).</i>	<i>inhibits(cdc20, cycA).</i>
<i>inhibits(cycA, cdh1).</i>	<i>inhibits(cycA, p27).</i>

$$\begin{array}{ll}
ab(p27, T). & \\
inh(p27, T) & \leftarrow cyclins\_active(cycA, cycE, T - 1), T > 0. \\
act(p27, T) & \leftarrow not\ cyclins\_active(cycA, cycE, T - 1), \\
& \quad active(p27, T - 1), T > 0. \\
cyclins\_active(cycA, cycE, T) & \leftarrow act(cycA, T), act(cycE, T).
\end{array}$$

time	CycD	Rb	E2F	CycE	CycA	p27	Cdc20	Cdh1	UbcH10	CycB
1	1	0	0	0	0	0	1	1	1	0
2	1	0	1	0	0	0	0	1	1	0
3	1	0	1	1	0	0	0	1	0	0
4	1	0	1	1	1	0	0	1	0	0
5	1	0	0	1	1	0	0	0	1	0
6	1	0	0	0	1	0	0	0	1	1
7	1	0	0	0	1	0	1	0	1	1
8	1	0	0	0	0	0	1	1	1	0

Table 6.4: Mammalian cell cycle execution flow

In Figure 6.6.8 the largest attraction basin of this network is shown and in Figure 6.6.9 an basin that attracts to a steady cycle is shown.

Table 6.4 represents a steady cycle characterized by the states at time points 1 and 8. This cycle almost literally corresponds to the original cycle reported by Faure et al. [49], with the only exception that *UbcH10* is expressed one step earlier (denoted with the bold font face in the table). This alteration, however, does not change the general flow of the process and thus can be considered a model-specific alteration.

## 6.7 Conclusions

In this chapter we have proposed modelling regulatory networks as answer set programs. Answer set programming is an area of logic programming that is capable of describing systems that exhibit non-monotone behaviour using negation-as-failure. These system models, represented as programs, can be executed to produce the set of steady states of a given protein interaction network.

We started with the development of an ASP framework that covers threshold boolean network semantics. The framework, described in Section 6.4.1 is a set of rules that expresses threshold boolean networks and analyzes their evolution over time. Besides the essential activation and inhibition link modelling, we introduce the concept of thresholds that characterize the susceptibility of a protein to activation or inhibition influence. Moreover, we explain how to use these notions for modelling phenomena of self-activation and self-inhibition and explain the subtle differences in the behaviour that can be achieved by different implementations of these phenomena.

Further, we extended this framework with the possibility to model ‘meta-interactions’, i.e., the influence of proteins on other interactions, not on other proteins, which allows for simpler more intuitive network descriptions. This functionality goes beyond standard boolean network semantics, since in Boolean networks one can only specify the interactions between proteins, but not between protein and interaction. Moreover, we augmented our framework with a flexible query mechanism that simplifies the analysis of constructed models. Queries can be used to compute only those trajectories in a state space that a biologist is interested in. A query is represented as an ASP program, which allows for building complex statements that may involve conditions and may evaluate different states of a network evolution trajectory. Finally, we devised an efficient algorithm that computes a complete state space of a network in a single solver run.



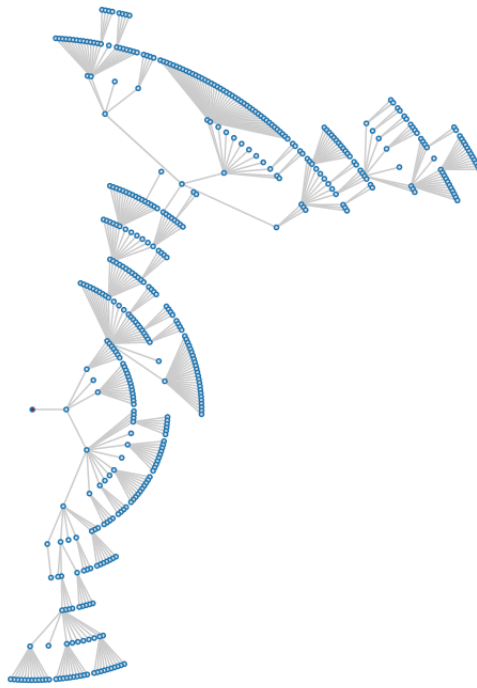


Figure 6.6.8: The biggest attraction basin of the Mammalian cell cycle network model

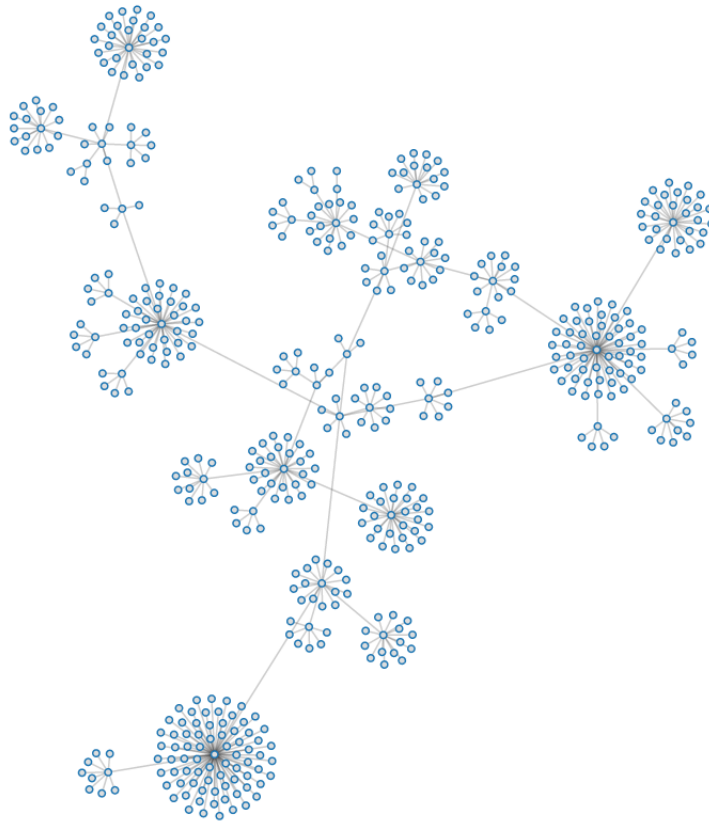


Figure 6.6.9: The attraction cycle of the Mammalian cell cycle network model

---

One of the main advantages of our ASP framework is that it is more formal compared to Boolean networks, since it allows a user to describe all implicit assumptions and background knowledge explicitly in the program, while in Boolean networks this knowledge often remains hidden in the non-formal description. This was illustrated in Sections 6.6.2 and 6.6.3, where the background information not explicitly expressed in the network description was represented with explicit rules in the ASP formulation. The approach remains however straightforward to apply; it does not require any formal logics knowledge from the biologist, who can operate with ready-to-apply blocks to build a model. At the same time the approach is very flexible due to the fact that nonmonotonic reasoning is used and any specific case which does not fit in the general picture can be incorporated with minimal effort. This was illustrated in Section 6.6.3, where the problem of expressing an interaction that does not fit in the framework semantics was solved by adding three simple rules.

Moreover, readily available answer set solvers can be used to find the steady states of a network. In Chapter 8 we present a software implementation of our framework using a third-party solver *clingo* [61].

# 7 Systems for gene and protein interaction retrieval and extraction

---

## 7.1 Introduction

Creating user interfaces and making text mining algorithms available for use to a broad audience is an important step for uniting computer science achievements with biological applications. In the last years an increasing amount of biological information has become available from different sources, ranging from highly structured databases of protein names, gene sequences and interactions such as BIND [11], Gene Ontology [6], UniProt [166], DIP [145] to name a few, to collections of loosely structured biomedical article abstracts such as HighWire [75] and MEDLINE [102], which create a rich ground for this synergy.

This thesis focuses on the specific problem of extracting and utilizing gene and protein interactions, thus in this chapter we will focus on information extraction and retrieval systems built to work with interactions. A typical scenario for interaction extraction or retrieval is when a user queries for a protein name and the system returns a structured representation of interactions of the query protein with other genes and proteins. The representation may range from a simple search engine-like list of interacting proteins to an interactive interaction graph. Online application architecture, which provides a single access point to various information sources, is an obvious choice for building such systems that provides a natural way of handling biomedical data. We provide an overview of the available online tools for interaction retrieval and extraction in Section 7.2. Given the substantial amount of existing online tools to explore the biomedical

literature for protein interactions, it is quite remarkable that we encountered two – at least from our perspective – obvious needs that have not been properly addressed so far:

- A unified framework for development and evaluation of PPI extraction methods.
- An extension of PPI extraction systems with modelling capabilities.

The development and evaluation framework we propose will address the needs of computer scientists that create new methods for PPI extraction and want to concentrate on this specific task, without spending much time on essential, but time-consuming preparatory work. On the other hand, the extension of PPI extraction systems with modelling capabilities will leverage biologists' experience with the ability to build models based on freshly extracted data. We discuss these two aspects in more detail in Section 7.3. Moreover, we address these needs by implementing a prototype of the system that includes these features in Chapter 8.

## 7.2 Online tools for biologists

Currently, most of the available online systems are focusing on the integration of different sources of structured biological information, while providing limited or no support for text mining algorithms. Systems that aim at handling unstructured data such as article abstracts mostly still rely on structured information sources such as gene and protein name lists or known interactions lists. These systems can be broadly divided into two classes: information retrieval (IR) and information extraction (IE) systems. Both types of systems are aimed at providing the user with information about an input query (e.g. the protein *Cdk1*); however the methodology and the output of these systems are different. IR systems typically operate on unstructured data such as text documents in order to find a subset of documents that are most relevant to the query. The output of such systems is typically a list of relevant documents that should be further processed by the user. IE systems on the other hand, use structured information sources such as databases and thesauri along with unstructured documents as well as sophisticated information processing algorithms to extract knowledge about a query concept. The output of such systems thus is not just a set of relevant documents, but is a structured representation of relationships between the query object (*Cdk1* in our case) and other entities, proteins, genes, organisms

where this protein appears, etc. Typical IE systems include an IR system as a component.

Table 7.1 provides an overview of systems that provide advanced information retrieval facilities for exploring the biomedical literature, which may already exploit some basic IE techniques such as statistical co-occurrence or manual rule matching. Table 7.2 describes systems that go beyond information retrieval, employ text mining approaches to extract information and provide more comprehensive means for exploring biomedical texts. Below we describe PubMed [126] as it is a prototypical IR system that is furthermore very often used as a component of IE systems. Further, we focus on the interaction extraction task and provide more details on IE-oriented systems.

PubMed is among the most popular IR systems for biologists. This is an information retrieval system that provides a web interface for MEDLINE [102], a database of references to articles in the life sciences. PubMed uses a vector space model, where every document is represented as a vector of words, and a weight is assigned to every element of the vector according to some frequency weighting scheme [98]. Every document vector in the collection is compared to a user query that is transformed to the vector representation as well, sorted according to the obtained relevance score and then presented to the user. Moreover, PubMed uses a query expansion technique [89] to provide a better result coverage. For example, if a user enters *yeast* as a query, this term is looked up in the UMLS [115] database and expanded with the term *Saccharomyces cerevisiae*, such that the abstracts that do not mention yeast but its Latin name will be included in the results as well.

PubGene [81] and STRING [80] are examples of IE systems that represent two of the most comprehensive tools that integrate a vast amount of biological information and data mining techniques to leverage biologists' research experience. However, while providing powerful algorithms for finding interactions based on structured information such as databases of curated interactions and gene sequences, these tools rely on relatively simple information extraction techniques such as co-occurrence based relation extraction and manual rules matching, and do not apply machine learning approaches when dealing with natural language texts.

PubGene maintains its own database of gene names and synonyms that are used to extract information about gene relations from the abstracts retrieved from PubMed. First, it tags the names of genes that occur in the abstracts, using a heuristic algorithm to resolve ambiguous gene names to corresponding gene identifiers. In the next step the frequency of co-occurrence for every pair of genes in every abstract is calculated, thus creating an association between two

Table 7.1: Online systems for advanced biomedical information retrieval

Method name	Description
EBIMed [136]	Uses a set of external biological databases and ontologies to annotate the entities that can be related to the user query in the retrieved abstracts.
GoPubMed [43]	Expands the user query and structures and reranks PubMed search results using Gene Ontology.
iHOP [76, 79]	An IR engine that connects the abstracts by hyperlinking them through gene and protein names.
Kleio [118]	An IR engine that implements an algorithm for gene and protein name disambiguation and acronym recognition.
LitMiner [39]	A tool that couples the functionality of a digital library with information extraction techniques based on manually built vocabularies and co-occurrence.
MedBlast [165]	Resolves a biological sequence to gene names and performs a search on PubMed.
ProLinks [13]	Uses a predefined vocabulary and statistical co-occurrence method to extract the relation between genes and proteins in abstracts.
PubMatrix [10]	A text analysis tool that employs a statistical co-occurrence metric to determine relationships between concepts in PubMed.
PubMed [126]	A state of the art information retrieval system for biologists.
PubReMiner [127]	Reranks and regroups PubMed search results by a number of criteria such as MESH terms, journal of publication etc., and provides word frequency statistics that facilitate building more efficient queries.
Textpresso [112]	Species-oriented full text IR engine enriched with a manually-defined ontology for semantic query expansion.
XplorMed [121]	An IR engine that groups the retrieved results by associated MESH terms and provides basic statistics for word importance in these abstracts.

Table 7.2: Online systems for biomedical information extraction

BioIE [42]	Uses a set of predefined templates to match specific relations between entities in abstracts.
ChiliBot [25]	A rule based information extraction system that extracts the relations between genes and proteins.
MEDIE [109]	An information extraction system that operates on syntactic sentence structure to answer queries.
PubGene [81]	A comprehensive tool that integrates many information sources into a single information extraction system.
STRING [80]	A tool that integrates many information sources and involves text analysis to answer user queries.

genes with the weight corresponding to the number of co-occurrences. This information is then used to construct literature-based gene networks.

STRING implements a more sophisticated interaction extraction algorithm than the one in PubGene, which includes not only statistical co-occurrence of gene and protein names in the abstracts but also involves natural language processing [146]. Here the authors use a rule-based approach implemented as a cascaded finite state automaton to annotate gene and protein names and to extract regulation and (de-)phosphorylation relations between them from the abstracts retrieved from PubMed and other sources (see [80] for more details). Similarly to PubGene, the authors build a list of gene and protein names using public databases but also expand it with orthographic variants. The input text is tagged with part-of-speech tags and the entities from the list are matched against this text. These entities are then expanded to accommodate the cases where the actual tagged entity is modified by adjectives, nouns or prepositional phrases. For example in the phrase *the ArcB sensory kinase*, *ArcB* is a tagged protein name, but the entity referred to in the sentence is described by the whole phrase. In the next step the annotated text is passed to another rule-based system that recognizes interaction verbs such as *phosphorylate*, *regulates*, *increase* and their grammatical forms in order to establish the relation between genes and proteins.

BioIE [42] proposes a more information retrieval-inspired approach that is based on matching a predefined set of templates and rules against a set of biomedical texts. These templates may be words and/or (not) contiguous phrases that are built manually to capture a certain relation of interest. The set of predefined patterns is aimed at detecting five categories of relations:



structure, function, diseases and therapeutic compounds and localization and familial relationships. These patterns can be extended by the user and also new patterns can be created to extend the functionality of the BioIE system.

ChiliBot [25], similarly to BioIE, is a rule-based interaction extraction system. Also, unlike in most other systems, in ChiliBot the user is expected to enter two (not one) protein names. In contrast to BioIE, ChiliBot employs natural language processing, namely parts-of-speech tagging and shallow parsing, to match the text against templates. These templates look for verb phrases between two query terms and for specific verbs within these phrases that define the relationship types. These relationships are divided into six categories: stimulatory, inhibitory, both stimulatory and inhibitory, neutral, parallel and co-occurrence only. The latter two are non-interactive categories that indicate that two genes or proteins only co-occur in the same sentence (parallel) or abstract (co-occurrence) but do not match any pattern. The system works online and thus the extracted information is limited to the 100 most recent relevant abstracts retrieved from PubMed to keep the response time acceptable.

MEDIE [109] is another information extraction system that employs more sophisticated text mining techniques to extract relations between genes and proteins. First, this approach preprocesses the whole MEDLINE database offline by tagging gene and protein names and related concepts with the help of external resources, and then parses these abstracts in order to obtain a syntactic structure for every sentence. The obtained structured information is then stored in a database, and user queries are transformed and looked up in the database online using a nested region algebra formalism [99].

PubGene, STRING and ChiliBot provide an additional functionality of constructing a network representation of gene and protein interactions. PubGene relies only on co-occurrence to build such networks and thus it does not indicate the direction or the nature of the relationship, while ChiliBot and STRING indicate both for certain relationship types.

## 7.3 Motivation for the development of PRISE

As we have discussed in Section 7.1 there are two aspect of PPI extraction process that have received little attention so far: the need for a unified development and evaluation framework, and the extension of conventional extraction systems with reasoning capabilities.

A unified framework is needed, because the current development of interaction extraction algorithms is complicated by the fact that a researcher has

to make a lot of effort in setting up the experimental environment in order to evaluate the value of his or her idea. Although conventional machine learning frameworks like Weka [72] can be used to facilitate the development and evaluation process of an interaction extraction algorithm, and some efforts are being made to unify the preparation of training and evaluation corpora [128], it still requires significant effort to set up an entire text mining pipeline that involves at least establishing the back-end data structure and implementing text database querying, NER and interaction extraction. This is especially frustrating if a researcher aims at contributing only to one part of the pipeline without delving too much in the other components. All systems described above do not explicitly share any implementation code or documentation, thus it is not possible to tell how much effort it would take to substitute any desired features.

The @Note framework [97] has been developed to address this problem. The goal of this framework is to facilitate the cooperation of biologists, data miners and computer scientists by combining a general purpose Natural Language Processing framework GATE [33] and a machine learning framework YALE [106] and to provide the extracted information in different perspectives for different types of users. However, although this framework provides excellent tools for annotation and overall textual data management, the tight integration with GATE NLP tools as well as limited documentation and rigid API limit the applicability of this framework for our purposes.

Another candidate framework for rapid data mining application development is the RapidMiner workbench [133]. This workbench follows the same path as @Note and encapsulates the YALE and Weka machine learning frameworks as well as many conventional modules that facilitate data pre- and post-processing. Although it does not specifically focus on the task of biomedical data mining like @Note, it provides a rich set of basic tools that facilitate fast development of custom modules. Moreover, the framework is well maintained and new releases are issued regularly.

To justify the need of information extraction and modelling integration, let us briefly review the paradigm of systems biology. The goal of systems biology is to build a reliable model that is consistent with the modelled organism, such that the perturbations in the organism and in the model cause the same responses. In order to build such a model, systems biology uses an iterative approach as described by Ideker et al. [77]:

1. Define all the components of the system.
2. Systematically perturb and monitor components of the system.
3. Reconcile the experimental results with those predicted by the model by

proposing model refinement hypotheses.

4. Design new perturbation experiments to distinguish between suggested model refinements and repeat steps 2-4.

All systems described above are concentrated on mining information and combining it together in a coherent model, i.e., performing step 1 in the algorithm above. On the other hand, as described in Chapter 4, there are a number of modelling approaches available that analyze biological models by means of executing them and observing the output, thus representing step 2 and in some cases step 3 of the algorithm above. However, to the best of our knowledge there is no system that combines these steps together in order to build a one-stop systems biology framework that seamlessly integrates information extraction and modelling. Such a framework would allow for better integration of existing knowledge in biological models and simplify the process of model development, thus potentially leading to more descriptive and accurate models.

In order to deal with the two major problems outlined in the beginning of this section we have implemented our own PRotein Interaction Search Engine (PRISE) – an open, configurable biomedical interaction extraction framework. To solve the first problem of providing an easy to use biomedical text mining testbed, with the arguments provided before in mind we chose RapidMiner as a development platform for PRISE as it is ideally suited for our purposes of constructing an easily extensible system. This choice leverages PRISE with all machine learning algorithms and data transformation methods that exist in RapidMiner and significantly reduces the amount of code that needs to be written from scratch.

To solve the second problem of providing a synergy between text mining and modelling, PRISE aims at making the first steps in uniting information extraction with biological model construction and analysis by implementing not only a relation extraction functionality but also providing a tool for building and analyzing the model from this data. A complete description of PRISE is provided in Chapter 8.

# 8

## PRISE - a PRotein Interaction Search Engine

---

### 8.1 Introduction

In Chapters 2 and 3 we have studied methods for protein-protein interaction (PPI) extraction. To briefly recall the details of the PPI extraction task, let us consider the following example.

**Example 1** Consider the sentence

In the *shaA* mutant,  $\sigma(H)$ -dependent expression of *spo0A* and *spoVG* at an early stage of sporulation was sensitive to external NaCl.

This sentence contains 4 protein names, namely *shaA*,  $\sigma(H)$ , *spo0A* and *spoVG*. The first step of the PPI extraction task, which is called Named Entity Recognition (NER), is to extract these names from the text. Assume that the 4 proteins in this sentence were successfully recognized by a NER method. These proteins can be combined into 6 unordered pairs, namely *shaA*- $\sigma(H)$ , *shaA*-*spo0A*, *shaA*-*spoVG*,  $\sigma(H)$ -*spo0A*,  $\sigma(H)$ -*spoVG*, and *spo0A*-*spoVG*. A protein pair is a positive instance if the original sentence expresses an interaction between the members of this pair, and a negative instance if they just co-occur in the sentence. In the example above, there are two positive instances, namely  $\sigma(H)$ -*spo0A* and  $\sigma(H)$ -*spoVG*, while the other 4 instances are negative. The second step in PPI extraction is to detect which pairs are positive instances; this phase is called PPID (protein-protein interaction detection).

More specifically, in Chapter 3 we were focusing on the PPID task, assuming that NER was performed in advance. In that chapter we studied the impact of different types of features extracted from text to represent a potential interaction on the accuracy of PPID methods. We studied lexical, shallow and deep syntactic features, and one interesting result of this study is that syntactic relations between words in a sentence, which represent a small subset of all induced features, allow obtaining a similar performance as a much more extended feature set that includes lexical data, i.e., words themselves.

Next, in Chapters 4, 5 and 6 we proposed an answer set programming (ASP) framework for regulatory network modelling. This framework extends the functionality of traditional Boolean networks with the ability to model ‘meta-interactions’, i.e., interactions that have other interactions as their targets. Moreover, in Chapter 6 we proposed a mechanism to answer queries about network states and trajectories as well as devised an efficient algorithm for finding network’s stable states.

In this chapter, we present a system that we have implemented along the way, i.e., while we carried out the research that led to the results presented in the previous chapters. It encompasses both the methodology for PPID which we studied in Chapter 3, as well as the answer set programming framework for regulatory networks modelling which we proposed in Chapter 6. In addition, this framework can be used to address the problems outlined in Chapter 7.

As we already mentioned in Chapter 7, it is desirable to have a unified development framework that simplifies the development and evaluation of PPI extraction methods. However, most of the methods developed for NER and PPID do not follow any standard specification, which means that they are implemented in different languages, with different data preprocessing techniques and for different platforms. Additionally, some researchers evaluate their systems on non-publicly available data sets, which makes it difficult to reproduce their experiments. Moreover, the NER and PPID components that are needed to build a full interaction extraction pipeline are often developed independently of each other in a research context and thus are not compatible. This also hampers the use of the developed methods in the real world, e.g. to build standalone applications containing the NER and PPID methods for the benefit of the biological community.

The chapter is divided in two parts. First, we present a framework that simplifies the development and evaluation process of PPI extraction methods. Second, we implement PRISE – a PRotein Interaction Search Engine that includes this framework, but also includes a reasoning module that incorporates the findings from Chapter 6.

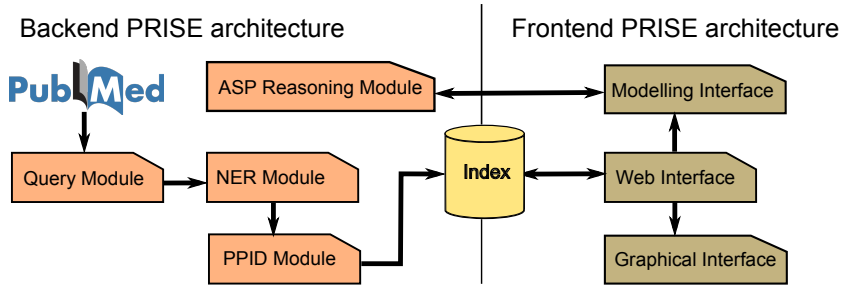


Figure 8.1.1: Component structure of PRISE

The first part is an attempt to take the first steps towards a flexible unified text mining framework for extracting protein-protein interactions. This framework aims to provide an easy way to:

1. build and evaluate PPI extraction systems, and to simplify the use of different data sets for training and testing purposes
2. build a stand-alone application server that is capable of processing and accumulating PPI data continuously, and disclosing this data to the user through a query mechanism

The implementation of this functionality is outlined in Section 8.3.

The second part, PRISE, is the application that summarizes the contributions of this thesis. The structure of this application is shown in Figure 8.1.1. Here one can see that the PRISE backend system actually constitutes a PPI extraction system that contains NER and PPID components as well as a separate reasoning module. The algorithms of the reasoning module are extensively described in Chapter 6, therefore we do not specifically focus on this module in the rest of the chapter (which does not mean that this module is not important). Most of the discussion in this chapter will focus the implementation of the PPI extraction pipeline and of the frontend part. As we already justified at the end of Chapter 7, we choose the RapidMiner machine learning framework as the basis for our implementation. To familiarize the reader with RapidMiner we provide an overview of the essential RapidMiner concepts in Section 8.2. Further in Section 8.3 we describe the structure of the implementation plugin, which is followed by a description of the PRISE frontend components in Section 8.4.

## 8.2 RapidMiner

RapidMiner is a machine learning framework that provides a modular view on machine learning processes, and can be used both as a standalone application and as a software library. For more information on RapidMiner we refer to [133].

The basic concepts of RapidMiner are `Operator` and `Process`. `Operator` and `OperatorChain` are terms used in RapidMiner to denote units that perform certain operations, such as reading data from a file, training a classifier, etc. An `OperatorChain` can contain other operators as its parts. A `Process` is a special case of an `OperatorChain` that is used to denote a set of operators aimed to solve a certain task, e.g. PPI extraction.

Every `Operator` has input and/or output ports that are used to receive and transmit data. Operators can be linked together on their ports to build a data processing pipeline, where each operator either transforms the input data or produces new data for a given input. For example, operator `Sample` may remove some entries from the input data set, while operator `k-NN` does not transform the input, but learns a model from it.

Typically, data in RapidMiner is stored in a specific data structure called `ExampleTable` that is connected to a persistent storage (database) containing all instances. The actual operators use `ExampleSets` which are views over an `ExampleTable` that may represent partial information from the corresponding `ExampleTable`. An `ExampleSet` consists of `Examples` that usually correspond to feature vectors of instances. In the standard RapidMiner framework, data are passed between operators only in the form of `ExampleSets`. However, if RapidMiner is used as a library it is possible to define other objects that can be passed between ports by inheriting the `IOObject` class. This is an important feature that we use in our framework as it allows us to build a domain model and operators that act directly on this model as we explain below.

## 8.3 Backend architecture

### 8.3.1 PPI extraction task

Supervised machine learning techniques can be used to address both NER and PPID. In general, for the NER task, a ‘window’ of several words around a candidate protein name or even a whole sentence is represented as a vector of features, such as words themselves, orthographic features (capitalization, hy-

phenation, alphanumeric characters), Greek letters, punctuation, etc. These feature vectors are further used to train a classifier based on a conventional machine learning technique. The results of NER can then be used to solve the PPID task.

To address the PPID problem, one needs to build representations for candidate interactions, i.e., for protein pairs. Such a protein pair representation should include information (features) from the sentence that can be used to distinguish between positive and negative instances. The difference with the NER task is that in PPID we already have a fixed pair of terms in a sentence and the task is not to detect an entity name, but to extract a (potentially implicit) relationship between these terms. This means that the strategy to build a feature vector for this task is likely to be different and potentially can include more elaborate features such as syntactic dependencies between words or other structural information. For an elaborated discussion on features that can be used for the PPID task we refer to our work described in Chapter 3.

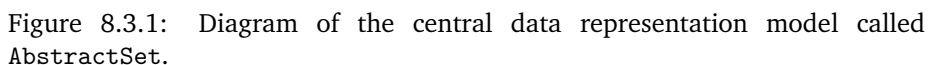
As we will see in Sections 8.3.3 and 8.3.4, a system to solve the PPI extraction problem can be decomposed into independent modules that generally fit well in the RapidMiner paradigm. The main difference between the NER and PPID tasks from the structural point of view is the way to obtain the feature vector representation of a candidate entity/interaction. Thus abstracting the operations that convert a sentence into a feature vector representation gives us a flexible framework that is suitable to build both NER and PPID components. To this end we have developed a set of additional RapidMiner operators that simplify this task and that provide extra functionalities specific to PPI extraction.

## 8.3.2 Domain models

### 8.3.2.1 Domain model structure

To better accommodate the PPI extraction domain we have developed our own data representation model shown in Figure 8.3.1 and called `AbstractSet`. It preserves the textual data on different granularity levels, rather than converting it to RapidMiner's `ExampleTable` immediately. In this structure, an abstract can contain 1 or more sentences; a sentence can contain 0 or more proteins and 0 or more interactions; an interaction involves exactly 2 proteins. The additional wrapper classes (with IO suffix) are intended to pass data between RapidMiner operators. The reason for introducing this representation is that RapidMiner assumes that every instance in the data set is represented as an





Example, but merely loading text in a RapidMiner Process does not provide us with a desirable feature vector. Moreover, as we do not want to restrict a user to any certain set of features, we want to provide a flexible mechanism for text to feature vector conversion that a user can implement depending on his needs, and the domain representation depicted in Figure 8.3.1 simplifies this task.

#### 8.3.2.2 PPI database

Another design choice is to use an external database to store the interaction related data. It is important to distinguish between RapidMiner's internal database used to store an `ExampleTable` and the database we use. In RapidMiner, a database is a medium to store the data set loaded in a `Process`, and this database is used to operate with data exclusively within this `Process`, i.e. it is a

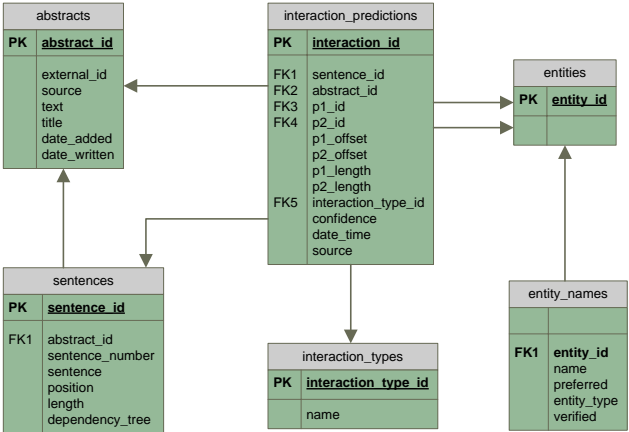


Figure 8.3.2: Entity-Relationship diagram of the PRISE database

transient storage.<sup>1</sup> In our case, to build a stand alone PPI extraction application, we need a persistent storage that contains the same information no matter what the Process’s structure is. The structure of this database is presented in Figure 8.3.2.

This database contains tables that store the information about the main data objects: abstracts, sentences, gene and protein names (table `entities`) and mined interactions. Abstracts can come from different sources (e.g. we can load annotated data sets in the same table that contains data from PubMed) and the field `source` stores the source of every abstract, while `external_id` stores the abstract’s id in this source. The field `date_written` contains the time stamp of writing the abstract to this table and the field `date_added` is a PubMed-specific field that contains the time stamp of adding this abstract to the PubMed database.

Every sentence in table `Sentences` has a link to a corresponding abstract, the number of this sentence in the abstract, the character offset of the first sentence character and a dependency tree for this sentence.

Tables `Entities` and `Entity_names` contain the information about gene and protein names. Two tables are used for this representation as one protein may

<sup>1</sup>[http://rapid-i.com/wiki/index.php?title=Data\\_core](http://rapid-i.com/wiki/index.php?title=Data_core)

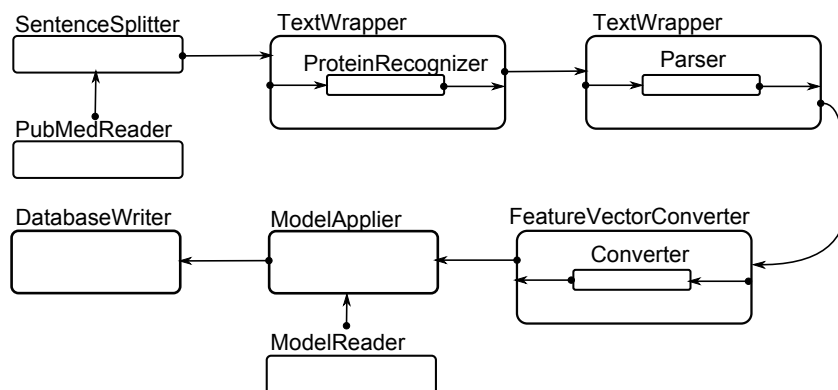


Figure 8.3.3: Diagram of a PPI extraction system that reads articles from MEDLINE (PubMed) and stores the mined interactions in a database. The labels correspond to RapidMiner operators.

have several names. For every name the type of entity is kept (gene or protein). The flag *preferred* indicates the canonical name for a given entity, the flag *verified* indicates that this name was checked manually and is a correct name.

Finally, table *Interaction\_predictions* contains all information that is necessary to identify an interaction. Every interaction is associated with a corresponding sentence, contains information about two entities that take part in it and contains the confidence score assigned by a PPID tool. The field *source* contains the information about the tool that has assigned the confidence score.

### 8.3.3 Application mode

The starting point for PPI extraction are articles from the MEDLINE text database. PubMed provides a query interface to this library that retrieves information from MEDLINE. To access this data we have built a *PubMedReader* operator that queries the online service with parameters such as keywords and date range, and provides a list of abstracts as output. These are stored in an *AbstractSet* as depicted in Figure 8.3.1. The initial text mining step to perform next is to split the text of these abstracts into sentences. To perform this preprocessing step one can use an existing sentence splitter that takes the text as input and outputs the list of sentences. To encapsulate this component we have built

a `SentenceSplitter` operator that takes an `AbstractSet` containing a list of `Abstract` objects as input and returns the same `AbstractSet` that additionally contains a list of `Sentence` objects for every abstract.

The next step in the application pipeline is to determine the entities (gene and protein names) which are potential interaction arguments. Here again one can use an existing NER tool that processes texts sentence by sentence and returns the sentences with annotated protein names. To this end we have developed a `RapidMiner` operator `TextWrapper` that has a parameter `Granularity` which regulates the granularity level of the iterator that reads the data (per abstract, per sentence or per (potential) interaction). This operator is connected to the `ProteinRecognizer` operator that encapsulates an external NER tool. `ProteinRecognizer` converts the format produced by the NER tool to the format of the data representation model we use, which now contains abstracts with sentences that contain annotated gene and protein names. If further syntactic preprocessing needs to be done, a natural language parser can be plugged in. Typically, parsers accept a sentence and output a parsing structure for this sentence; in this case a parser can also be encapsulated with `TextWrapper` to ensure correct input/output processing. For a discussion of the actual tools used in our implementation we refer to Section 8.3.5.

For the PPID step we do not follow the ‘wrapping’ strategy we used for NER and sentence splitting. Instead, we offer the user the ability to take advantage of readily available classifiers implemented in `RapidMiner`, as well as provide him with a mechanism to prepare arbitrary feature vectors as was discussed in Section 8.3.1. To this end we need to support the conversion of a candidate interaction to a feature vector representation. Note that typically more features beyond mere terms are induced from the text for complex information extraction tasks. The induced features may range from n-grams, lemmas, POS tags to more complex ones such as syntactic roles and dependency trees. To perform this conversion we have developed a `FeatureVectorConverter` operator chain that takes the domain data structure as input and processes it interaction by interaction in its inner chain, converting every instance to an `Example` object that represents a feature vector for this instance. The operator `Converter` in the inner chain that performs the actual conversion should be implemented by the user. In the application mode we assume that the classifier was trained elsewhere (see Section 8.3.4) and thus it is being read by a standard `ModelReader` operator.

To complete the picture we have implemented a `DatabaseWriter` operator that provides access to the underlying interaction database. The input of this operator is an `AbstractSet` that is being written to the database.

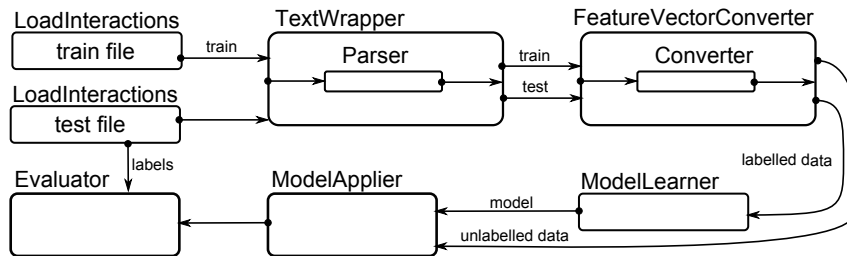


Figure 8.3.4: Diagram of the evaluation of a PPID method on an annotated benchmark data set.

### 8.3.4 Evaluation mode

Our RapidMiner framework for PPI extraction can be used for evaluation purposes too. In this section we discuss an evaluation pipeline for PPID as shown in Figure 8.3.4; evaluation of NER follows a similar path.

Instead of pulling in unseen articles from MEDLINE, in the evaluation mode one typically has annotated data that is used both for training and evaluation. One of the problems with annotated data sets is that they can come in different formats. Recently, a unified XML format for PPID was proposed [129]. We have implemented a `LoadInteractions` operator that reads such an XML file and produces an `AbstractSet` as output. Alternatively, a user can read the interactions from the domain database using the operator `DatabaseReader` or implement an operator that converts a proprietary data format to an `AbstractSet` for further processing. Typically, PPID benchmark data sets contain texts that are split into sentences with interactions, gene and protein names already annotated. This means that the output of this operator is a fully populated `AbstractSet`, as opposed to the application case in Section 8.3.3 where several preprocessing steps were needed to arrive at this stage. The subsequent feature vector construction step is the same as in Section 8.3.3.

The next steps involve training and evaluating a classifier. RapidMiner provides a number of popular classification algorithms that can be readily applied through learner operators, as well as allows for inclusion of new algorithms by implementing custom learner operators for them. In Figure 8.3.4 we abstract from any particular algorithm and denote the learner as `ModelLearner`. The input to a learner is an `ExampleSet` and the output is a RapidMiner `Model` that stores the parameters of the model learned by a classifier. This model can be

further applied to classify new instances by means of the `ModelApplier` operator. The `Evaluator` operator takes predictions from the `ModelApplier` along with actual labels of instances and computes various evaluation metrics such as accuracy, and classification error values.

### 8.3.5 PRISE backend implementation

We have used the RapidMiner framework described above to build PRISE - a system that scans PubMed entries and provides a web interface for querying the extracted interactions. To build the application pipeline we have used an implementation of a sentence splitter provided in the OpenNLP<sup>2</sup> framework and a NER tool called ABNER [150].

For PPID, we used the combined kernel as explained in Chapter 3. However, dependency trees are not in our domain model, thus we needed to include an extra step to parse the sentences. We implemented a `Parser` operator that wraps the Stanford Parser<sup>3</sup> and enriches our data structure with parse and dependency trees, which potentially allow us to extract not only shortest paths but also POS tags, chunks, phrases etc. To build a classifier we used a `TreeSVM` implementation [111] of the `SVMLight` classifier [82]. `TreeSVM` takes the trees in the Penn Treebank notation as input, thus the `Converter` operator merely extracts the shortest path from the sentence dependency tree for a given protein pair and returns an `Example` that contains a shortest path in the Penn Treebank notation as a single feature. Further, the `Converter` in a straightforward way extends the feature vector with different types of shortest paths such as those that contain POS as nodes and those that contain only syntactic dependencies as described in Chapter 3.

Note that having a tree as a feature vector generally poses a problem in machine learning frameworks such as Weka [72] as trees are difficult to fit into the concept of feature vectors implemented there. However, in the modular framework built with RapidMiner we isolate this problem in one component, namely the classifier itself, and keep the rest of the pipeline untouched, which means that we can switch to another application setup by merely changing the `Converter` operator and the classifier.

So far, we restricted our attention to the interaction extraction pipeline, and we did not say anything about the ASP reasoning module depicted in Figure 8.1.1, although it is a very important part of PRISE. The implementation of

---

<sup>2</sup><http://opennlp.sourceforge.net/>

<sup>3</sup><http://nlp.stanford.edu/software/lex-parser.shtml>

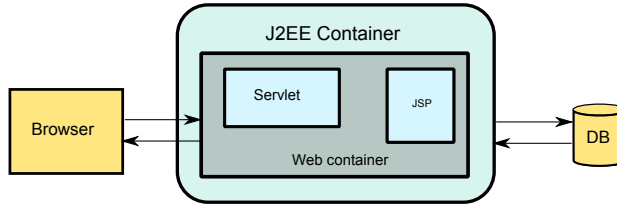


Figure 8.4.1: The general architecture of a J2EE web container

this module is, in fact, a straightforward implementation of the SSB algorithm described in Chapter 6, Section 6.5.1 and of the extended Temporal algorithm described in the same chapter, Section 6.5.2. To solve ASP programs we have used the Clingo package [61] that provides a grounder and a solver under a common hood.

## 8.4 Frontend architecture

### 8.4.1 Server-side architecture

The frontend component should provide online access to the data, gathered by the backend component. A natural choice for online frontend systems is to use the Java EE (J2EE) framework that provides a formal specification of web server interfaces. The advantage of using J2EE is that many publicly available implementations of this specification such as Tomcat<sup>4</sup> or GlassFish<sup>5</sup> can be used to facilitate the application development. Moreover, there exist frameworks that provide templates for typical web application needs such as Apache Struts<sup>6</sup> or Spring<sup>7</sup> that allow a developer to focus only on the implementation of the application parts that are essential to the system. In this section we briefly outline the typical web application architecture and emphasize the choices we made for our application.

The structure of a J2EE web container is shown in Figure 8.4.1. A client, represented as a web browser, sends a request to a web server that implements

<sup>4</sup><http://tomcat.apache.org/>

<sup>5</sup><https://glassfish.dev.java.net/>

<sup>6</sup><http://struts.apache.org/>

<sup>7</sup><http://www.springsource.org/>

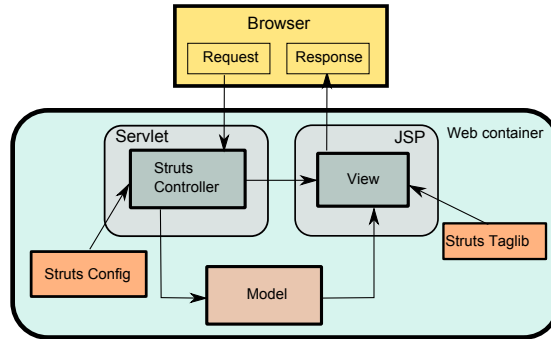


Figure 8.4.2: Struts application architecture

the J2EE web container specification. According to the specification the request is transmitted to a servlet, which decides which response to a client should be generated. Java Server Pages (JSP) are often used to model templates for different response types. The response generation process is typically associated with requesting domain data which can be queried from a database or some other source. Finally, the generated response page is returned to the web container, which in turn returns the response to a browser.

This specification follows the MVC (Model-View-Controller) architectural principle, that decouples the application into three components, each of which can be developed independently. **Model** represents the application logic and typically consists of data structures that are populated from databases and transformed according to the application needs. In our case the model is presented by the database depicted in Figure 8.3.2 and the domain data structure depicted in Figure 8.3.1. **View** is used to represent the model to a user and allows a user to interact with it. In our case the view is presented as the PRISE web interface. Finally, **controller** connects the view and the model by defining the interaction logic between them.

The MVC pattern is a best practice to use when developing web applications, thus to simplify the development we have chosen to use the Apache Struts framework that supports development compliant with this pattern. Struts enforces the MVC pattern on J2EE web application development by providing an extension of the servlet container and facilitating the development of view components. The architecture of a Struts application is shown in Figure 8.4.2. As



---

cluster	The action used to query the interactions for a given entity and to represent the results as a list. Interaction evidences are grouped per second interacting entity.
graph	The action used to query the interactions for a graphical representation. Retrieved interactions are returned in XML format and further processed in a browser.
xml	The action to update the existing graphical representation. See Section 8.4.2 for more details.
model	The action that runs the modelling algorithm described in Chapter 6.

---

Table 8.1: Description of Struts actions implemented in PRISE

can be seen from Figure 8.4.2, Struts is built on top of the J2EE web container. It provides an extension of the servlet class that is governed by the Struts configuration file. Requests from a browser come as ‘actions’ defined in the configuration file. Each action has an associated controller, that interacts with the model. Each action has a set of associated views, one of which is returned depending on the outcome of the controller execution. A library of Struts tags that extends the conventional JSP tag library is used to eventually shape the response to a browser. The list of actions implemented in the system with short explanations is provided in Table 8.1.

### 8.4.2 Client-side architecture

The web interface should allow a user to

1. query the interactions for a given gene or protein from the database and represent the results in textual and graphical form,
2. expand the existing query results with interactions for other proteins,
3. filter the results,
4. edit interaction networks obtained by multiple queries,
5. run network modelling algorithms that compute the network’s steady states.

The textual representation of search results can be straightforwardly implemented with plain HTML, while the graphical representation poses a problem in a browser setting, as web browsers are poorly suited to represent graphical interactive content. To overcome this problem many extensions were proposed to build interactive browser applications such as Flash, ActiveX, Java Applets, etc. However, most of these technologies either require to install additional software and/or are not cross-platform, i.e. they do not work in all browsers. With these arguments in mind we opted for the HTML5 *canvas* model that allows a programmer to create interactive graphics within a browser window. The fact that it is a part of the HTML5 specification makes it work on most modern browsers. Moreover, for advanced visualization of network state spaces we have used the Protovis library <http://vis.stanford.edu/protovis/>.

The problem with the incremental query feature is that the browser should store already obtained results when sending the request for new interactions and processing the response. Initially, browsers were designed to be stateless, i.e. when they receive a new portion of data they ‘forget’ everything that they had before. However, in many cases it is useful to store the information obtained earlier, i.e. to keep the state of the browser. The standard way to implement this functionality is to use the AJAX (Asynchronous JavaScript and XML) technology that sends requests and receives responses without reloading the full page. To implement the incremental update in our application, when extending the query results we call a special action `xml` (see Table 8.1), which returns a response that is recognized by the browser as an incremental update, and thus it does not reload the page.

Using the canvas model we can represent the interactions retrieved from the database as a graph, as shown in Figure 8.4.3. The nodes represent entities and the links between them represent the interactions mined from abstracts. The thickness of the links reflects the number of citations that support this interaction - a thicker line means more abstracts describe this interaction. The color of the link represents the confidence of a PPID method in this interaction - darker lines represent more certain interactions.

The results can be incrementally extended, as shown in Figure 8.4.4. To this end, a user can click on a node already presented in the interaction graph and click the button ‘Search for interactions’. This will expand the displayed network with the interactions for the new query protein. The result of query expansion is shown in Figure 8.4.4, where the query proteins are depicted with red nodes.

### PRISE graphical search and network editing

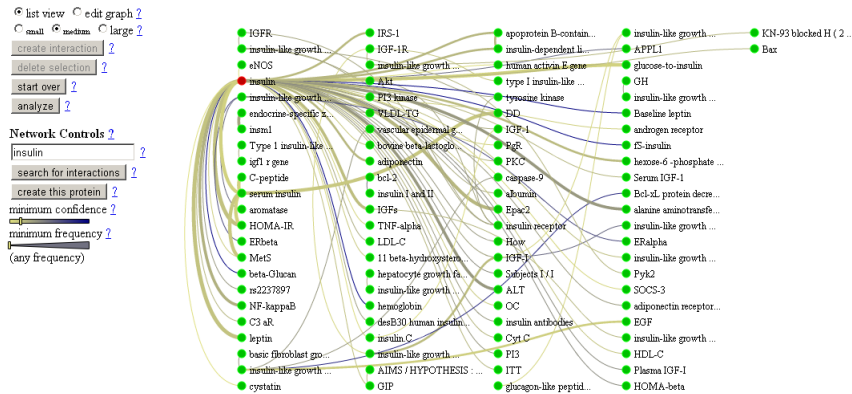


Figure 8.4.3: Graphical representation of search results

### PRISE graphical search and network editing

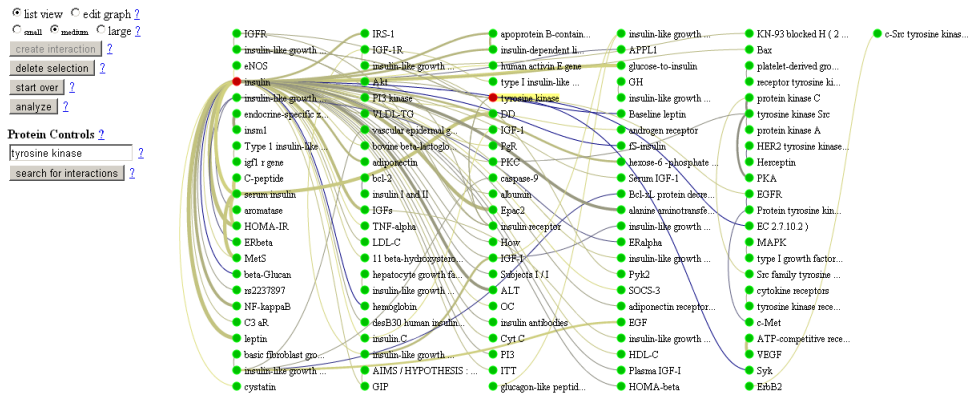


Figure 8.4.4: Incremental search results

### PRISE graphical search and network editing

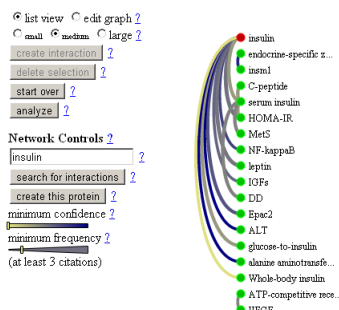


Figure 8.4.5: Filtered results

Additional controls shown on the left can be used to filter the results. Two sliders filter the results by frequency and by classifier confidence. Figure 8.4.5 depicts the interactions from the Figure 8.4.4 that have at least 3 supporting evidences.

Further, with the radio button in the top left corner we can switch to the editing mode where the results are represented as a network that can be edited and analyzed. The editing screen is presented in Figure 8.4.6. In this figure, the interaction between insulin and leptin is selected, and on the left panel the user can see the links to the abstracts that mention this interaction along with the confidence score assigned by a PPID method.

Moreover, the application provides a way to build interaction networks from scratch, as shown in Figure 8.4.7. Control buttons create new proteins and interactions between them, set activation and inhibition thresholds and express the exceptions to the framework semantics. Moreover, it is possible to enhance edges with references linking to a relevant paper on PubMed.

In Figure 8.4.7 the Fission yeast cell cycle network model is depicted. This model can be analyzed with our ASP framework by clicking the ‘Analyze’ button. The state space of a network is represented on the following screen, depicted in Figure 8.4.8. This screen allows a user to browse through attraction basins of the modelled network. At every moment, one attraction basin is shown, and the user can switch between the basins by means of a radio button on the left. Labels describing the state associated with every node can be switched on and off as well.

### PRISE graphical search and network editing

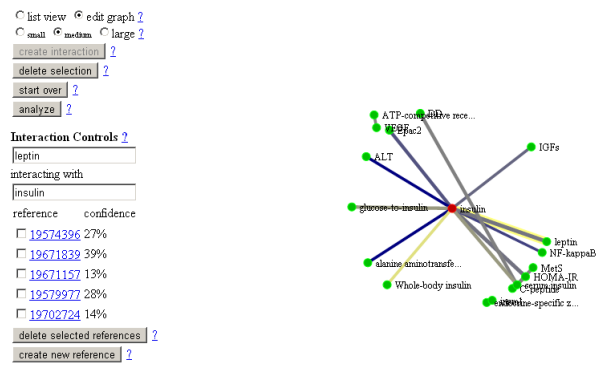


Figure 8.4.6: Editing mode

### PRISE graphical search and network editing

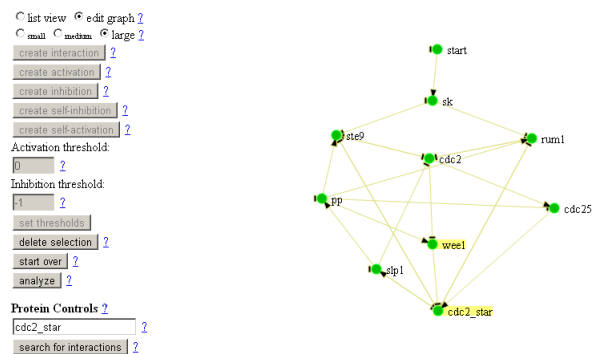


Figure 8.4.7: Editing mode

**PRISE: state space diagram**

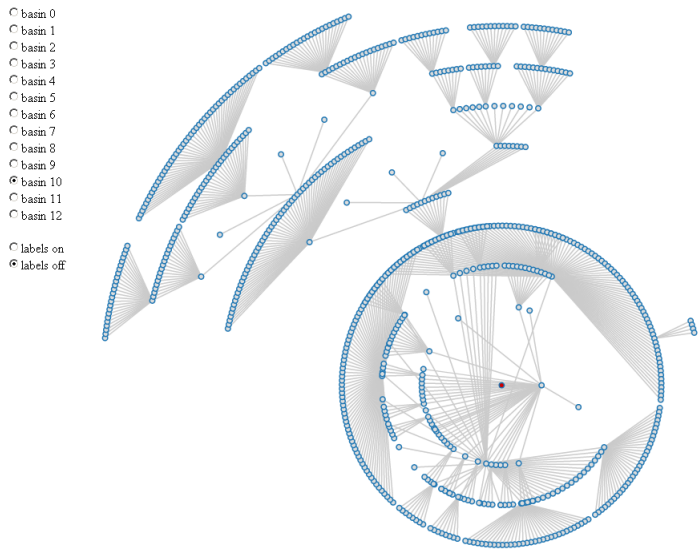


Figure 8.4.8: Editing mode

# 9

## Summary

---

The advances in biological research in the last two decades have provided many new insights on the functioning principles of living organisms. This resulted in the proposal of new methods to study the development of living organisms. One of the most significant advances is the development of microarrays and of high-throughput techniques that allow a biologist to collect a massive amount of experimental information about a particular biological process, which can be used for different purposes such as explanation of a living organism behaviour or drug design.

The development of new experimental methods and rapid growth of biological data repositories have contributed to building biological network models that are aimed at reconstructing real biological processes *in silico*. On the other hand, advances in experimental techniques have led to a rapid increase of reports that describe new biological discoveries. Clearly, processing large amounts of information calls for automated methods that (partially) remove the burden of manual data processing from the shoulders of biologists.

These two aspects of effective biological data utilization, namely automated text processing for information extraction and constructing and analyzing biological network models, have attracted a lot of attention in computer science. In this thesis we have presented our contributions to these problems by studying both information extraction and modelling aspects.

In the first part of the thesis, that comprises Chapters 2 and 3, we have discussed the problem of information extraction from biological article abstracts, and in particular the problem of protein-protein interaction extraction. In Chapter 2 we have introduced the basic notions necessary to perform the interaction extraction task; we have discussed the natural language processing techniques

and the machine learning setting for this task. Further, we have described the annotated corpora used for machine learning training and evaluation. These explanations not only provided a solid foundation for understanding the methods described in Chapter 3, but were also useful for the implementation part of our work, presented in Chapter 8.

In Chapter 3 we have studied the impact of different feature types on the performance of protein-protein interaction extraction methods. To this end we have examined different structured kernels with Support Vector Machines. We have taken four kernels that reflect different degrees of using syntactic and lexical information and have performed three types of experiments to study the behaviour of these methods under different conditions on five benchmark datasets. The most important observation in this chapter is that by using only grammatical relations we can obtain a similar performance as with an extended feature set. This indicates the relative importance of grammatical information for the interaction extraction task. Another finding is the correlation between training/test set sizes and the method choice. We observed that when the training set is much smaller than the test set, then the syntactic kernel performs better. We believe that these findings can be helpful in building faster and less complicated classifiers, as well as for choosing a proper kernel according to the data set at hand.

The second part of this thesis consists of Chapters 4, 5 and 6. These chapters address the problem of biological system modelling. In the last decade this problem has attracted a lot of attention from the theoretical computer science field; in Chapter 4 we have discussed some of the computational modelling techniques that have been proposed as possible solutions. As we have discussed in this chapter, there is still no general agreement on the tools that suit the needs for any particular modelling task. Some approaches, such as Petri Nets, are best suited for modelling metabolic networks since they provide a natural representation for production/consumption processes, while others, such as Boolean networks, are more suitable for gene regulatory network modelling, since gene states are easily approximated by boolean states. One of the problems that is common for all approaches is that it is not clear how easy it is to integrate new information in the model, and whether it can be done in a (semi) automatic way, thus leveraging the model construction with data and text mining techniques that extract gene and protein interactions from the otherwise unused data.

In Chapters 5 and 6 we have described a new approach to regulatory network modelling based on Answer Set Programming (ASP) that combines an easy network description language with powerful means for network analysis.



In particular, in Chapter 5 we have introduced Markovian answer set programs, which are useful to model systems whose behaviour depends on time. An important task when modelling such systems is to find their steady states and cycles. A brute force approach for steady state finding may lead to a bad solving time. To overcome this problem we have proposed an effective algorithm for solving Markovian programs.

Next, in Chapter 6 we have proposed modelling regulatory networks as Markovian programs. We started with the development of an ASP framework that covers threshold boolean network semantics. Besides the essential activation and inhibition link modelling, we introduced the concept of thresholds that characterize the susceptibility of a protein to activation or inhibition influence, and we explained how to use these notions for modelling self-activation and self-inhibition phenomena. Further, we extended this framework with the possibility to model ‘meta-interactions’, i.e., the influence of proteins on other interactions, which goes beyond the reach of conventional boolean network semantics. Moreover, we augmented our framework with a flexible query mechanism that simplifies the analysis of constructed models. Queries can be used to compute only those trajectories in a state space that a biologist is interested in. A query is represented as an ASP program, which allows for building complex statements that may involve conditions and may evaluate different states of a network evolution trajectory. Finally, we devised a specific algorithm that computes a complete state space of a network in a single solver run.

As we discussed further in Chapter 6, one of the main advantages of our ASP framework is that it is more formal compared to Boolean networks, since it describes all implicit assumptions and background knowledge explicitly in the program, while in Boolean networks this knowledge often remains hidden in the non-formal description. However, the approach remains straightforward to apply; it does not require any formal logics knowledge from the biologist, who can operate with ready-to-apply blocks to build a model. At the same time the approach is very flexible due to the fact that nonmonotonic reasoning is used and any specific case which does not fit in the general picture can be incorporated with minimal effort. Finally, in the end of Chapter 6 we have provided three regulatory network models for Budding Yeast, Fission Yeast and the mammalian cell cycle, that illustrate the features of our modelling framework.

The third part of this thesis consists of Chapters 7 and 8. This part draws upon the research that we have performed in the previous chapters and describes the software implementation of the information extraction methods from Chapter 3 and of the modelling algorithms from Chapter 6. Chapter 7 provides an overview of online information retrieval and extraction systems that have

been built to address the information extraction problem in the biological domain. Further, we discussed the need for a development and evaluation framework for interaction extraction. Such a framework can eliminate the need of building a whole text mining pipeline from scratch, which would allow a researcher to concentrate on the actual extraction task. Finally, in Chapter 8 we have proposed an implementation of such a framework that is built on top of the RapidMiner machine learning library. Moreover, in this chapter we have presented PRISE – a PRotein Interaction Search Engine that integrates this framework. Furthermore, PRISE implements the modelling algorithms from Chapter 6 and provides a web tool to build and analyze regulatory networks.

The problem of biological data processing keeps attracting a lot of interest from the computer science domain, and the state of the art in information extraction and reasoning is constantly pushed forward by improving upon previous developments and proposing new approaches. One interesting path for extending the work presented in this dissertation is to extend our modelling framework with the possibility to model gradual activation. Other aspects that can be studied include asynchronous execution and enrichment of the framework with different interaction types beyond activation and inhibition.





# Bibliography

- [1] AIROLA, A., PYYSALO, S., BJÖRNE, J., PAHIKKALA, T., GINTER, F., AND SALAKOSKI, T. A graph kernel for protein-protein interaction extraction. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing* (2008).
- [2] ALBERT, R. Boolean modeling of genetic regulatory networks. *Lecture Notes in Physics* 650 (2004), 459–481.
- [3] ALBERT, R., AND OTHMER, H. G. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in drosophila melanogaster. *Journal of Theoretical Biology* 223, 1 (2003), 1 – 18.
- [4] AMARAL, L. A. N., DIAZ-GUILERA, A., MOREIRA, A. A., GOLDBERGER, A. L., AND LIPSITZ, L. A. Emergence of complex dynamics in a simple model of signaling networks. *Proceedings of the National Academy of Sciences* 101, 44 (2004), 15551–15555.
- [5] ANANIADOU, S., AND MCNAUGHT, J. *Text Mining for Biology And Biomedicine*. Arttech House, 2005.
- [6] ASHBURNER, M., BALL, C. A., BLAKE, J. A., BOTSTEIN, D., BUTLER, H., CHERRY, J. M., DAVIS, A. P., DOLINSKI, K., DWIGHT, S. S., EPPIG, J. T., HARRIS, M. A., HILL, D. P., ISSEL-TARVER, L., KASARSKIS, A., LEWIS, S., MATESE, J. C., RICHARDSON, J. E., RINGWALD, M., RUBIN, G. M., AND SHERLOCK, G. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature genetics* 25, 1 (2000), 25–29.
- [7] BALDUCCINI, M., GELFOND, M., AND NOGUEIRA, M. Answer set based design of knowledge systems. *Annals of Mathematics and Artificial Intelligence* 47, 1-2 (2006), 183–219.

- [8] BARAL, C. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
- [9] BARAL, C., CHANCELLOR, K., TRAN, N., TRAN, N., JOY, A. M., AND BERENS, M. E. A knowledge based approach for representing and reasoning about signaling networks. *Bioinformatics* 20, 1 (2004), 15–22.
- [10] BECKER, K., HOSACK, D., DENNIS, G., LEMPICKI, R., BRIGHT, T., CHEADLE, C., AND ENGEL, J. Pubmatrix: a tool for multiplex literature mining. *BMC Bioinformatics* 4, 1 (2003), 61.
- [11] <http://bond.unleashedinformatics.com/>.
- [12] BOSER, B. E., GUYON, I., AND VAPNIK, V. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the 5th annual workshop on Computational learning theory* (1992), pp. 144–152.
- [13] BOWERS, P., PELLEGRINI, M., THOMPSON, M., FIERRO, J., YEATES, T., AND EISENBERG, D. Prolinks: a database of protein functional linkages derived from coevolution. *Genome Biology* 5, 5 (2004), R35.
- [14] BRAND, M. D. Regulation analysis of energy metabolism. *Journal of Experimental Biology* 200, 2 (1997), 193–202.
- [15] BRANTS, T. Tnt - a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing* (2000), pp. 224–231.
- [16] BREWKA, G. *Nonmonotonic reasoning: logical foundations of common sense*. Cambridge University Press, New York, NY, USA, 1991.
- [17] BRILL, E. Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. *Computational Linguistics* 21, 4 (1995), 543–565.
- [18] BUNESCU, R., GE, R., KATE, R. J., MARCOTTE, E. M., MOONEY, R. J., RAMANI, A. K., AND WONG, Y. W. Comparative experiments on learning information extractors for proteins and their interactions. *Artificial Intelligence in Medicine* 33, 2 (2005), 139–155.
- [19] BUNESCU, R. C., AND MOONEY, R. J. Subsequence kernels for relation extraction. In *Proceedings of 19th Annual Conference on Neural Information Processing Systems* (2005).
- [20] BUSI, N., AND GORRIERI, R. A petri net semantics for pi-calculus. In *Proceedings of the 6th International Conference on Concurrency Theory* (1995).

- 
- [21] CER, D., DE MARNEFFE, M.-C., JURAFSKY, D., AND MANNING, C. D. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *Proceedings of the Conference on Language Resources, Technologies and Evaluation* (2010).
  - [22] CHAOUIYA, C. Petri net modelling of biological networks. *Briefings in Bioinformatics* 8, 4 (2007), 210–219.
  - [23] CHAOUIYA, C., REMY, E., RUET, P., AND THIEFFRY, D. Qualitative modelling of genetic networks: From logical regulatory graphs to standard petri nets. In *Proceedings of the 25th International Conference on Applications and Theory of Petri Nets* (2004), pp. 137–156.
  - [24] CHAOUIYA, C., REMY, E., AND THIEFFRY, D. Qualitative petri net modelling of genetic networks. In *Transactions on Computation Systems Biology* (2006), pp. 95–112.
  - [25] CHEN, H., AND SHARP, B. Content-rich biological network constructed by mining pubmed abstracts. *BMC Bioinformatics* 5, 1 (2004), 147.
  - [26] CHEUNG, D., AND SONG, I.-Y., Eds. *CIKM '09: Proceedings of the 18th ACM conference on Information and knowledge management* (New York, NY, USA, 2009), ACM.
  - [27] CIOCCHETTA, F., AND HILLSTON, J. Process algebras in systems biology. In *Formal Methods for Computational Systems Biology* (2008).
  - [28] CLEGG, A., AND SHEPHERD, A. Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics* 8, 1 (2007), 24.
  - [29] COLLINS, M., AND DUFFY, N. Convolution kernels for natural language. In *Proceedings of 14th Annual Conference on Advances in Neural Information Processing Systems* (2001), pp. 625–632.
  - [30] CORTES, C., AND VAPNIK, V. Support-vector networks. *Machine Learning* (1995), 273–297.
  - [31] CRESTANI, F., MARCHAND-MAILLET, S., CHEN, H.-H., EFTHIMIADIS, E. N., AND SAVOY, J., Eds. *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010* (2010), ACM.
  - [32] CRICK, T., BRAIN, M., VOS, M., AND FITCH, J. Generating optimal code using answer set programming. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning* (2009).

- [33] CUNNINGHAM, H., MAYNARD, D., BONTCHEVA, K., AND TABLAN, V. Gate: A framework and graphical development environment for robust nlp tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics* (2002).
- [34] CURTI, M., DEGANI, P., PRIAMI, C., AND BALDARI, C. T. Modelling biochemical pathways through enhanced pi-calculus. *Theoretical Computer Science* 325, 1 (2004), 111–140.
- [35] DAVIDICH, M. I., AND BORNHOLDT, S. Boolean network model predicts cell cycle sequence of fission yeast. *PLoS ONE* 3, 2 (2008), e1672.
- [36] DAVIDSON, E., AND LEVIN, M. Gene regulatory networks. *Proceedings of the National Academy of Sciences of the United States of America* 102, 14 (2005), 4935–4935.
- [37] DAVIS, J., AND GOADRIC, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning* (2006).
- [38] DE JONG, H. Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology* 9, 1 (2002), 67–103.
- [39] DEMAINE, J., MARTIN, J., WEI, L., AND DE BRUIJN, B. Litminer: integration of library services within a bio-informatics application. *Biomedical Digital Libraries* 3, 1 (2006), 11.
- [40] DEVILLERS, R. R., KLAUDEL, H., AND KOUTNY, M. A petri net translation of  $\pi$ -calculus terms. In *Proceedings of 3rd International Colloquium on Theoretical Aspects of Computing* (2006).
- [41] DING, J., BERLEANT, D., NETTLETON, D., AND WURTELE, E. S. Mining medline: Abstracts, sentences, or phrases? In *Pacific Symposium on Biocomputing* (2002).
- [42] DIVOLI, A., AND ATTWOOD, T. K. Bioie: extracting informative sentences from the biomedical literature. *Bioinformatics* 21, 9 (2005), 2138–2139.
- [43] DOMS, A., AND SCHROEDER, M. GoPubMed: exploring PubMed with the Gene Ontology. *Nucleic Acids Research* 33, suppl.2 (2005), W783–786.
- [44] DWORSCHAK, S., GRELL, S., NIKIFOROVA, V. J., SCHAUB, T., AND SELBIG, J. Modeling biological networks by action languages via asp. *Constraints* 13, 1-2 (2008), 21–65.
- [45] EKER, S., KNAPP, M., LADEROUTE, K., LINCOLN, P., MESEGUER, J., AND SÖNMEZ, M. K. Pathway logic: Symbolic analysis of biological signaling. In *Pacific Symposium on Biocomputing* (2002).



- 
- [46] EKER, S., KNAPP, M., LADEROUTE, K., LINCOLN, P., AND TALCOTT, C. L. Pathway logic: Executable models of biological networks. *Electronic Notes in Theoretical Computer Science* 71 (2002).
  - [47] EOM, J.-H., AND ZHANG, B.-T. Pubminer: Machine learning-based text mining for biomedical information analysis. *Genomics & Informatics* 2, 2 (2004), 99–106.
  - [48] FAN, R.-E., CHEN, P.-H., AND LIN, C.-J. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research* 6 (2005), 1889–1918.
  - [49] FAURE, A., NALDI, A., CHAOUIYA, C., AND THIEFFRY, D. Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle. *Bioinformatics* 22, 14 (2006), e124–131.
  - [50] FAYRUZOV, T., DE COCK, M., CORNELIS, C., AND HOSTE, V. Deeper: A full parsing based approach to protein relation extraction. In *Proceedings of EvoBIO08* (2008), pp. 36–47.
  - [51] FAYRUZOV, T., DE COCK, M., CORNELIS, C., AND HOSTE, V. The role of syntactic features in protein interaction extraction. In *Proceedings of ACM Second International Workshop of Data and Text Mining Methods in Bioinformatics* (2008), pp. 61–68.
  - [52] FAYRUZOV, T., DE COCK, M., CORNELIS, C., AND HOSTE, V. Linguistic feature analysis for protein interaction extraction. *BMC Bioinformatics* 10, 1 (2009), 374.
  - [53] FAYRUZOV, T., DE COCK, M., CORNELIS, C., AND VERMEIR, D. Modeling protein interaction networks with answer set programming. In *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine* (2009).
  - [54] FAYRUZOV, T., DITTMAR, G., SPENCE, N., DE COCK, M., AND TEREDESAI, A. A rapidminer framework for protein interaction extraction. In *RapidMiner community meeting* (2010).
  - [55] FAYRUZOV, T., JANSSEN, J., VERMEIR, D., CORNELIS, C., AND DE COCK, M. Efficient solving of time-dependent answer set programs. In *Technical Communications of the 26th International Conference on Logic Programming* (2010), pp. 64–73.
  - [56] FAYRUZOV, T., JANSSEN, J., VERMEIR, D., CORNELIS, C., AND DE COCK, M. Modelling gene and protein regulatory networks with answer set

- programming. *International Journal of Data Mining and Bioinformatics* (In press).
- [57] FISHER, J., AND HENZINGER, T. A. Executable cell biology. *Nature Biotechnology* 25, 11 (2007), 1239–1249.
- [58] FUNDEL, K., KÜFFNER, R., AND ZIMMER, R. Relex - relation extraction using dependency parse trees. *Bioinformatics* 23, 3 (2007), 365–371.
- [59] GEBSER, M., GUZIOŁOWSKI, C., IVANCHEV, M., SCHAUB, T., SIEGEL, A., THIELE, S., AND VEBER, P. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning* (2010).
- [60] GEBSER, M., KAMINSKI, R., KAUFMANN, B., OSTROWSKI, M., SCHAUB, T., AND THIELE, S. Engineering an incremental asp solver. In *Proceedings of the 24th International Conference on Logic Programming* (2008).
- [61] GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. *clasp* : A conflict-driven answer set solver. In *Proceedings of the 9th International conference on Logic Programming and Nonmonotonic Reasoning* (2007).
- [62] GELFOND, M., AND LIFSCHITZ, V. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming* (1988), pp. 1070–1080.
- [63] GELFOND, M., AND LIFSCHITZ, V. Representing actions in extended logic programming. In *Proceedings of the Joint International Conference and Symposium on Logic Programming* (1992).
- [64] GELFOND, M., AND LIFSCHITZ, V. Representing action and change by logic programs. *Journal of Logic Programming* 17 (1993), 301–322.
- [65] GIMENEZ, J., AND MRQUEZ, L. Svmtool: A general pos tagger generator based on support vector machines. In *In Proceedings of the 4th International Conference on Language Resources and Evaluation* (2004).
- [66] GIULIANO, C., LAVELLI, A., AND ROMANO, L. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *11th Conference of the European Chapter of the Association for Computational Linguistics* (2006).
- [67] GOSS, P. J. E., AND PECCOUD, J. Quantitative modeling of stochastic systems in molecular biology by using stochastic petri nets. *Proceedings of the National Academy of Sciences of the United States of America* 95, 12 (1998), 6750–6755.

- 
- [68] GREFENSTETTE, G., AND TAPANAINEN, P. *What is a word, what is a sentence? Problems of tokenization*, 1994.
- [69] GRINBERG, D., LAFFERTY, J., AND SLEATOR, D. A robust parsing algorithm for link grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies* (1995).
- [70] GUPTA, S., BISHT, S. S., KUKRETI, R., JAIN, S., AND BRAHMACHARI, S. K. Boolean network analysis of a neurotransmitter signaling pathway. *Journal of Theoretical Biology* 244, 3 (2007), 463 – 469.
- [71] HALÁCSY, P., KORNAI, A., AND ORAVECZ, C. Hunpos: an open source trigram tagger. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions* (2007).
- [72] HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. The weka data mining software: an update. *SIGKDD Explor. Newsl.* 11, 1 (2009), 10–18.
- [73] HAUSSLER, D. Convolution kernels on discrete structures. Tech. rep., 1999.
- [74] HEARST, M. A. Untangling text data mining. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (1999).
- [75] <http://highwire.stanford.edu/>.
- [76] HOFFMANN, R., AND VALENCIA, A. A gene network for navigating the literature. *Nature Genetics* 36 (2004), 664.
- [77] IDEKER, T., GALITSKI, T., AND HOOD, L. A new approach to decoding life: systems biology. *Annual review of genomics and human genetics* 2, 1 (2001), 343–372.
- [78] IELPA, S. M., IIRITANO, S., LEONE, N., AND RICCA, F. An asp-based system for e-tourism. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning* (2009).
- [79] <http://www.ihop-net.org/>.
- [80] JENSEN, L. J., 0004, M. K., STARK, M., CHAFFRON, S., CREEVEY, C., MULLER, J., DOERKS, T., JULIEN, P., ROTH, A., SIMONOVIC, M., BORK, P., AND VON MERING, C. String 8 - a global view on proteins and their functional interactions in 630 organisms. *Nucleic Acids Research* 37, Database-Issue (2009), 412–416.

- [81] JENSSEN, T.-K., LÆGREID, A., KOMOROWSKI, J., AND HOVIG, E. A literature network of human genes for high-throughput analysis of gene expression. *Nature Genetics* 28, 21-28 (2001).
- [82] JOACHIMS, T. Making large-scale support vector machine learning practical. *Advances in kernel methods: support vector learning* (1999), 169–184.
- [83] JOACHIMS, T. *Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
- [84] KATRENKO, S., AND ADRIAANS, P. Learning relations from biomedical corpora using dependency tree levels. In *Proceedings of the Fifteenth Dutch-Belgian Conference on Machine Learning* (2006).
- [85] KAUFFMAN, S. A. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology* 22, 3 (1969), 437–467.
- [86] KESHAVA PRASAD, T. S., GOEL, R., KANDASAMY, K., KEERTHIKUMAR, S., KUMAR, S., MATHIVANAN, S., TELIKICHERLA, D., RAJU, R., SHAFREEN, B., VENUGOPAL, A., BALAKRISHNAN, L., MARIMUTHU, A., BANERJEE, S., SOMANATHAN, D. S., SEBASTIAN, A., RANI, S., RAY, S., HARRYS KISHORE, C. J., KANTH, S., AHMED, M., KASHYAP, M. K., MOHMOOD, R., RAMACHANDRA, Y. L., KRISHNA, V., RAHIMAN, B. A., MOHAN, S., RANGANATHAN, P., RAMABADRAN, S., CHAERKADY, R., AND PANDEY, A. Human protein reference database–2009 update. *Nucleic Acids Research* 37, suppl.1 (2009), D767–772.
- [87] KIM, S., YOON, J., AND YANG, J. Kernel approaches for genic interaction extraction. *Bioinformatics* 24, 1 (2008), 118–126.
- [88] KLEIN, D., AND MANNING, C. D. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics* (2003).
- [89] KNECHT, L. W. S., AND NELSON, S. J. Mapping in pubmed. *Journal of the Medical Library Association* 90, 4 (2002), 475.
- [90] KÖSTER, M., NOVÁK, P., MAINZER, D., AND FUHRMANN, B. Two case studies for jazzyk bsm. In *Agents for Games and Simulations: Trends in Techniques, Concepts and Design* (2009).

- 
- [91] KUHN, H. W., AND TUCKER, A. W. Nonlinear programming. In *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probability* (1950), J. Neyman, Ed., pp. 481–492.
- [92] LEASE, M., AND CHARNIAK, E. Parsing biomedical literature. In *Proceedings of the Second International Joint Conference on Natural Language Processing (IJCNLP-05), Jeju Island, Korea* (2005).
- [93] LECCA, P., PRIAMI, C., LAUDANNA, C., AND CONSTANTIN, G. A biospi model of lymphocyte-endothelial interactions in inflamed brain venules. In *Pacific Symposium on Biocomputing* (2004).
- [94] LI, F., LONG, T., LU, Y., OUYANG, Q., AND TANG, C. The yeast cell-cycle network is robustly designed. *PNAS* 101 (2004), 4781–4786.
- [95] LIFSCHITZ, V., AND TURNER, H. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming* (1994), pp. 23–37.
- [96] LODHI, H., TAYLOR, J. S., CRISTIANINI, N., AND WATKINS, C. J. C. H. Text classification using string kernels. In *Proceedings of 14th Annual Conference on Advances in Neural Information Processing Systems* (2000).
- [97] LOURENO, A., CARREIRA, R., CARNEIRO, S., MAIA, P., GLEZ-PEA, D., FDEZ-RIVEROLA, F., FERREIRA, E. C., ROCHA, I., AND ROCHA, M. @note: A workbench for biomedical text mining. *Journal of Biomedical Informatics* 42, 4 (2009), 710 – 720.
- [98] MANNING, C. D., RAGHAVAN, P., AND SCHÜTZE, H. *Introduction to Information Retrieval*, 1 ed. Cambridge University Press, July 2008.
- [99] MASUDA, K., AND TSUJII, J. Nested region algebra extended with variables for tag-annotated text search. In *Proceeding of the 17th ACM conference on Information and knowledge management* (2008).
- [100] MAURIN, M., MAGNIN, M., AND ROUX, O. H. Modeling of genetic regulatory network in stochastic pi-calculus. In *BICoB* (2009), pp. 282–294.
- [101] McDONALD, R., LERMAN, K., AND PEREIRA, F. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning* (2006).
- [102] [http://www.nlm.nih.gov/databases/databases\\_medline.html](http://www.nlm.nih.gov/databases/databases_medline.html).
- [103] MENDOZA, L., THIEFFRY, D., AND ALVAREZ-BUYLLA, E. R. Genetic control of flower morphogenesis in arabidopsis thaliana: a logical analysis. *Bioinformatics* 15, 7 (1999), 593–606.

- [104] MERCER, J. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London* 209 (1909), 415–446.
- [105] MESEGUER, J. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science* 96, 1 (1992), 73–155.
- [106] MIERSWA, I., WURST, M., KLINKENBERG, R., SCHOLZ, M., AND EULER, T. Yale: rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006).
- [107] MILNER, R. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
- [108] MITCHELL, T. M. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.
- [109] MIYAO, Y., OHTA, T., MASUDA, K., TSURUOKA, Y., YOSHIDA, K., NINOMIYA, T., AND TSUJII, J. Semantic retrieval for the accurate identification of relational concepts in massive textbases. In *Proceedings of the 44th annual meeting of the ACL* (2006).
- [110] MONTANARI, U., AND PISTORE, M. Concurrent semantics for the pi-calculus. *Electronic Notes Theoretical Computer Science* 1 (1995).
- [111] MOSCHITTI, A. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of the 17th European Conference on Machine Learning* (2006).
- [112] MÜLLER, H.-M., KENNY, E. E., AND STERNBERG, P. W. Textpresso: An ontology-based information retrieval and extraction system for biological literature. *PLoS Biol* 2, 11 (09 2004), e309.
- [113] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77, 4 (apr 1989), 541–580.
- [114] NEDELLEC, C. Learning language in logic - genic interaction extraction challenge. In *Proceedings of the ICML-2005 Workshop on Learning Language in Logic* (2005).
- [115] NELSON, S. J., POWELL, T., AND HUMPHREYS, B. L. The unified medical language system (umls) project. In *Encyclopedia of Library and Information Science*, A. Kent and C. M. Hall, Eds. Marcel Dekker Inc., New York, 2002, pp. 369–378.

- 
- [116] NIEMELÄ, I. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25, 3-4 (1999), 241–273.
  - [117] NIVRE, J. Non-projective dependency parsing in expected linear time. In *In Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics* (2009).
  - [118] NOBATA, C., COTTER, P., OKAZAKI, N., REA, B., SASAKI, Y., TSURUOKA, Y., TSUJII, J., AND ANANIADOU, S. Kleio: a knowledge-enriched information retrieval system for biology. In *Proceedings of the 31st Annual International ACM SIGIR Conference* (2008).
  - [119] OHTA, T., TATEISI, Y., AND KIM, J.-D. The genia corpus: an annotated research abstract corpus in molecular biology domain. In *Proceedings of the second international conference on Human Language Technology Research* (2002), pp. 82–86.
  - [120] PALMER, D. D., AND HEARST, M. A. Adaptive multilingual sentence boundary disambiguation. *Computational Linguistics* 23, 2 (1997), 241–267.
  - [121] PEREZ-IRATXETA, C., BORK, P., AND ANDRADE, M. A. Xplormed: a tool for exploring medline abstracts. *Trends in Biochemical Sciences* 26, 9 (2001), 573 – 575.
  - [122] PETERSON, J. L. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.
  - [123] PORTER, M. F. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.
  - [124] PRIAMI, C., MERELLI, E., GONZALEZ, P. P., AND OMICINI, A., Eds. *Transactions on Computational Systems Biology III* (2005), vol. 3737 of *Lecture Notes in Computer Science*, Springer.
  - [125] PRIAMI, C., REGEV, A., SHAPIRO, E., AND SILVERMAN, W. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters* 80, 1 (2001), 25–31.
  - [126] <http://www.ncbi.nlm.nih.gov/pmc/>.
  - [127] <http://bioinfo.amc.uva.nl/human-genetics/pubreminer/>.
  - [128] PYYSALO, S., AIROLA, A., HEIMONEN, J., BJORNE, J., GINTER, F., AND SALAKOSKI, T. Comparative analysis of five protein-protein interaction corpora. *BMC Bioinformatics* 9, Suppl 3 (2008), S6.

- [129] PYYSALO, S., AIROLA, A., HEIMONEN, J., BJORNE, J., GINTER, F., AND SALAKOSKI, T. Comparative analysis of five protein-protein interaction corpora. *BMC Bioinformatics, special issue 9*, Suppl 3 (2008), S6.
- [130] PYYSALO, S., GINTER, F., HAVERINEN, K., HEIMONEN, J., AND SALAKOSKI, T. On the unification of syntactic annotations under the stanford dependency scheme: A case study on bioinfer and genia. In *Proceedings of the BioNLP Conference* (2007).
- [131] PYYSALO, S., GINTER, F., HEIMONEN, J., BJÖRNE, J., BOBERG, J., JÄRVINEN, J., AND SALAKOSKI, T. BioInfer: A corpus for information extraction in the biomedical domain. *BMC Bioinformatics* 8 (2007), 50.
- [132] RAPE, M., AND KIRSCHNER, M. W. Autonomous regulation of the anaphase-promoting complex couples mitosis to s-phase entry. *Nature* 432 (2004), 588 – 595.
- [133] <http://rapid-i.com/>.
- [134] RAY, O. Nonmonotonic abductive inductive learning. *Journal of Applied Logic* 7, 3 (2009), 329–340.
- [135] RAY, O., WHELAN, K. E., AND KING, R. D. A nonmonotonic logical approach for modelling and revising metabolic networks. In *Proceedings of the 2nd International Workshop on Computational Intelligence in Security for Information Systems* (2009).
- [136] REBHOLZ-SCHUHMAN, D., KIRSCH, H., ARREGUI, M., GAUDAN, S., RYNBEEK, M., AND STOEHR, P. Protein annotation by ebimed. *Nature Biotechnology* 24 (2006), 902–903.
- [137] REDDY, V. N., LIEBMAN, M. N., AND MAVROVOUNIOTIS, M. L. Qualitative analysis of biochemical reaction systems. *Computers in Biology and Medicine* 26, 1 (1996), 9 – 24.
- [138] REGEV, A., SILVERMAN, W., AND SHAPIRO, E. Y. Representation and simulation of biochemical processes using the pi-calculus process algebra. *Pacific Symposium on Biocomputing 2001* (2001), 459–470.
- [139] REYNAR, J. C., AND RATNAPARKHI, A. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing* (1997).
- [140] ROHLF, T., AND BORNHOLDT, S. Criticality in random threshold networks: annealed approximation and beyond. *Physica A: Statistical Mechanics and its Applications* 310, 1-2 (2002), 245 – 259.



- 
- [141] ROSEN, R. *Essays on Life itself*. Columbia University Press, 1999.
  - [142] SAETRE, R., SAGAE, K., AND TSUJII, J. Syntactic features for protein-protein interaction extraction. In *Proceedings of the 3rd International Symposium on Languages in Biology and Medicine (Short Papers)* (2007).
  - [143] SAEYS, Y., INZA, I. N., AND LARRAÑAGA, P. A review of feature selection techniques in bioinformatics. *Bioinformatics* 23, 19 (2007), 2507–2517.
  - [144] SAGAE, K., MIYAO, Y., AND TSUJII, J. Hpsg parsing with shallow dependency constraints. In *Proceedings of the 45th Annual Meeting of the ACL* (2007).
  - [145] SALWINSKI, L., MILLER, C. S., SMITH, A. J., PETTIT, F. K., BOWIE, J. U., AND EISENBERG, D. The database of interacting proteins: 2004 update. *Nucleic Acids Research* 32, suppl.1 (2004), D449–451.
  - [146] SARIC, J., JENSEN, L. J., OUZOUNOVA, R., ROJAS, I., AND BORK, P. Extraction of regulatory gene/protein networks from medline. *Bioinformatics* 22, 6 (2006), 645–650.
  - [147] SCHOLKOPF, B., AND SMOLA, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.
  - [148] SCHONBACH, C. Molecular biology of protein-protein interactions for computer scientists. In *Biological data mining in protein interaction networks* (2009).
  - [149] SERRA, R., VILLANI, M., GRAUDENZI, A., AND KAUFFMAN, S. Why a simple model of genetic regulatory networks describes the distribution of avalanches in gene expression data. *Journal of Theoretical Biology* 246, 3 (2007), 449 – 460.
  - [150] SETTLES, B. Abner: an open source tool for automatically tagging genes, proteins and other entity names in text. *Bioinformatics* 21, 14 (2005), 3191–3192.
  - [151] SIEGEL, A., RADULESCU, O., BORGNE, M. L., VEBER, P., OUY, J., AND LAGARRIGUE, S. Qualitative analysis of the relation between dna microarray data and behavioral models of regulation networks. *Biosystems* 84, 2 (2006), 153 – 174.
  - [152] SIMONS, P., NIEMELÄ, I., AND SOININEN, T. Extending and implementing the stable model semantics. *Artificial Intelligence* 138, 1-2 (2002), 181–234.

- [153] SNOUSSI, E. H., AND THOMAS, R. Logical identification of all steady states : the concept of feedback loop characteristic states. *Bulletin of Mathematical Biology*, 55 (1993), 973–991.
- [154] SOYER, O. S., SALATH, M., AND BONHOEFFER, S. Signal transduction networks: Topology, response and biochemical processes. *Journal of Theoretical Biology* 238, 2 (2006), 416 – 425.
- [155] SUNDHEIM, B., AND GRISHMAN, R., Eds. *Proceedings of the 6th Conference on Message Understanding, MUC 1995, Columbia, Maryland, USA, November 6-8, 1995* (1995).
- [156] TALCOTT, C. Symbolic modeling of signal transduction in pathway logic. In *Proceedings of the 2006 Winter Simulation Conference* (2006).
- [157] TALCOTT, C. Pathway logic. In *Formal Methods for Computational Systems Biology* (2008), vol. 5016 of LNCS, Springer, pp. 21–53.
- [158] THIEFFRY, D., AND THOMAS, R. Dynamical behaviour of biological regulatory networksii. immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology* 57, 2 (1995), 277–297.
- [159] THOMAS, R., THIEFFRY, D., AND KAUFMAN, M. Dynamical behaviour of biological regulatory networksi. biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bulletin of Mathematical Biology* 57, 2 (1995), 247–276.
- [160] TIKK, D., THOMAS, P., PALAGA, P., HAKENBERG, J., AND LESER, U. A comprehensive benchmark of kernel methods to extract protein-protein interactions from literature. *PLoS Comput Biol* 6, 7 (July 2010), e1000837+.
- [161] TIWARI, A., AND TALCOTT, C. L. Analyzing a discrete model of aplysia central pattern generator. In *Proceedings of the 6th Conference on Computational Methods in Systems Biology* (2008).
- [162] TOUTANOVA, K., KLEIN, D., MANNING, C. D., AND SINGER, Y. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology* (2003).
- [163] TRAN, N. *Reasoning and hypothesing about signaling networks*. PhD thesis, Arizona State University, December 2006.
- [164] TSURUOKA, Y., AND TSUJII, J. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the conference*

- 
- on Human Language Technology and Empirical Methods in Natural Language Processing* (2005).
- [165] TU, Q., TANG, H., AND DING, D. MedBlast: searching articles related to a biological sequence. *Bioinformatics* 20, 1 (2004), 75–77.
- [166] <http://www.uniprot.org/>.
- [167] VAN KLINKEN, J. B. *The Modelling and Analysis of Dynamic Biochemical Systems – an integrative approach*. PhD thesis, University of Siena, February 2009.
- [168] VAN LANDEGHEM, S., ABEEL, T., SAEYS, Y., AND VAN DE PEER, Y. Discriminative and informative features for biomolecular text mining with ensemble feature selection. *Bioinformatics (in press)* (2010).
- [169] VAN LANDEGHEM, S., SAEYS, Y., DE BAETS, B., AND VAN DE PEER, Y. Extracting protein-protein interactions from text using rich feature vectors and feature selection. *Proceedings of Third International Symposium on Semantic Mining in Biomedicine* (2008).
- [170] VITERBI, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory* 13, 2 (2003), 260–269.
- [171] XIAO, J., SU, J., ZHOU, G., AND TAN, C. Protein-protein interaction extraction: A supervised learning approach. In *Proceedings of the 1st International Symposium on Semantic Mining in Biomedicine* (2005).
- [172] YAKUSHIJI, A., MIYAO, Y., OHTA, T., TATEISI, Y., AND TSUJII, J. Automatic construction of predicate-argument structure patterns for biomedical information extraction. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing* (2006).

